

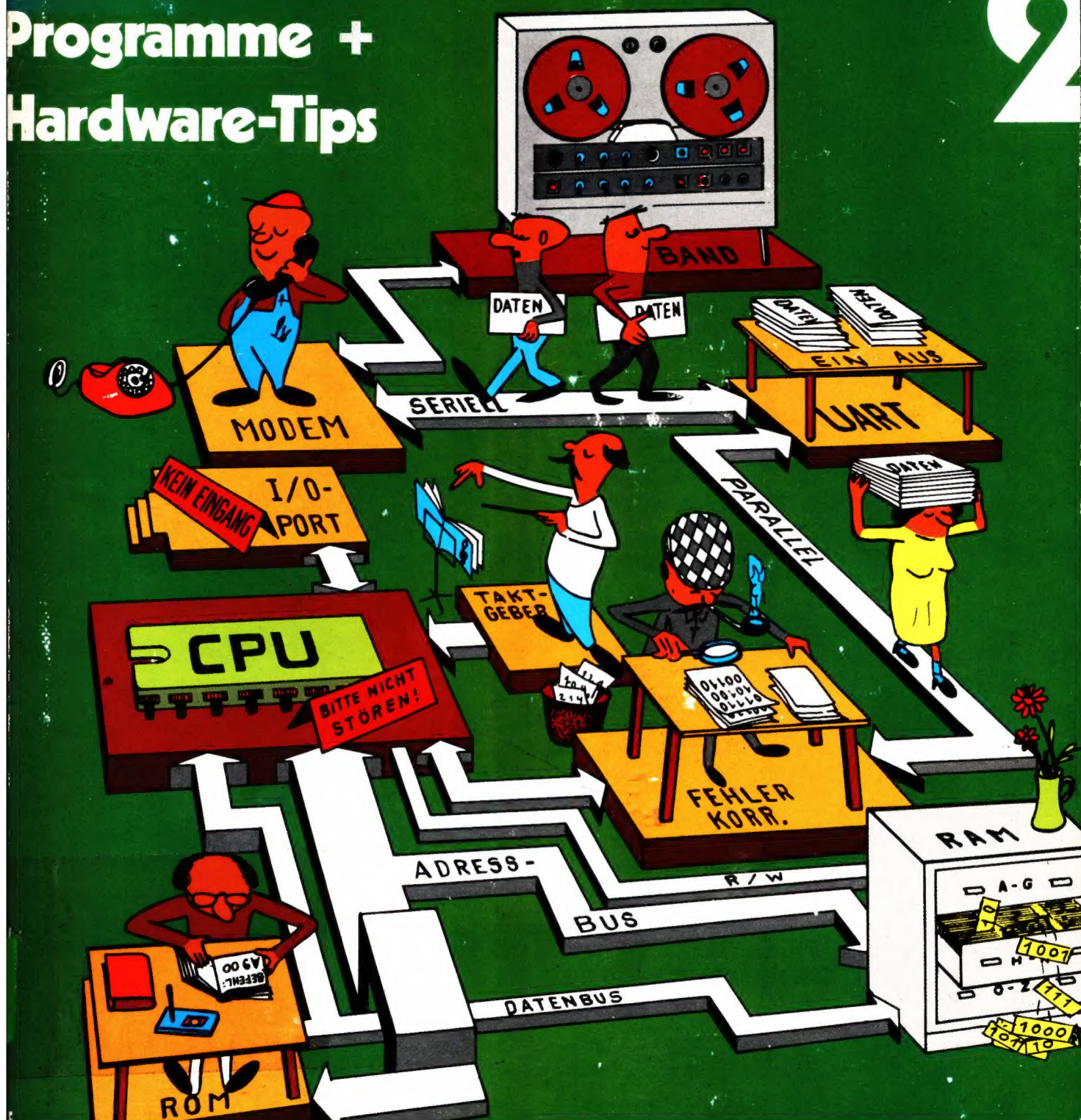
# HOBBI COMPUTER

Sonderheft der  
**Funkschau**

Preis DM 14,-

Programme +  
Hardware-Tips

2





# Vorwort

Als vor einem knappen Jahr das erste HOBBYCOMPUTER-Sonderheft des Franzis-Verlages erschien, hatte der Siegeszug der „Personal Computer“ gerade erst begonnen. Heute hat man zwar noch kein besseres Wort für die immer preiswerter gewordenen Mikrocomputer in Heim und Büro gefunden – auch „Hobbycomputer“ ist nicht ganz korrekt –, doch wurde für viele tausend Mitbürger in unserem Land aus der grauen Theorie inzwischen handfeste Praxis, wie auch die Verkaufszahlen der Hersteller beweisen.

Das nun vorliegende zweite HOBBYCOMPUTER-Sonderheft möchte jenen, die bereits einen Mikrocomputer besitzen, durch zahlreiche Programmierbeispiele und Hardware-Beiträge eine effektive Ausnutzung ihres Systems erleichtern, und jenen, die sich mit dem Gedanken der Anschaffung eines Computers tragen, eine Hilfestellung beim Vergleich der heute üblichen Mikroprozessoren geben. Weitere Beiträge befassen sich mit solchen Dingen wie Datenübertragung und Fehlerkorrektur – Dinge, über die man nur relativ wenig Literatur findet, und die doch eine zunehmende Rolle für Mikrocomputer-Anwender spielen.

Die Beiträge in diesem Heft sind exklusiv, d.h. sie sind bisher nirgendwo sonst erschienen. Daß sie bestimmte Mikroprozessoren und Mikrocomputer-Systeme ein wenig bevorzugen, liegt in erster Linie an deren Verbreitung auf dem Hobby-Sektor; trotzdem können meist auch die Benutzer anderer Systeme aus den abgedruckten Programmen Nutzen ziehen.

*Die Redaktion*



In nebenstehendem Inhaltsverzeichnis findet sich ziemlich alles wieder, was hier – etwas stilisiert – Bestandteil eines Mikrocomputer-Systems ist: Mit ihrem bisherigen System Unzufriedene können eine „Super-CPU“ mit individuellen Eigenschaften simulieren, mehrere Beiträge befassen sich mit der Verbindung des Computers zur Außenwelt, und ausführliche Berichte sind der Übertragung von Daten gewidmet.

1979

Franzis-Verlag GmbH, Karlstr. 37, 8000 München 2.  
 Produktion: ELVAG, Elektronik Verlag Luzern AG, CH-6002 Luzern. Bearbeitet von der Redaktion der Zeitschrift FUNKSCHAU. Für den Text verantwortlich: Herwig Feichtinger. Sämtliche Rechte – besonders das Übersetzungsrecht – an Text und Bildern vorbehalten. Fotomechanische Vervielfältigung nur mit Genehmigung des Verlages. Jeder Nachdruck, auch auszugsweise, und jede Wiedergabe der Abbildungen, auch in verändertem Zustand, sind verboten.

Druck: C. J. Bucher AG, CH-6002 Luzern

Grundlagen

Hardware

Software

Kurzbeiträge



# Inhalt

Bits und Bytes – eine kleine Einführung in die Maschinenprogrammierung .....	4	Zahlendarstellung im PET .....	19
Der beste Mikroprozessor – Überlegungen zur Systemauswahl .....	15	Effiziente BASIC-Programmierung .....	23
Vom Parity-Bit zur Kreuzparität .....	17	Wanzenjagd – Über die Fehlersuche in BASIC-Programmen .....	25

Datenübertragung per Telefon .....	30	Ein preiswerter Drucker für den TRS-80 .....	43
Ein Netzteil für den AIM-65 .....	32	Der TRS-80 und die Außenwelt .....	47
Das VIA 6522 – ein intelligenter Peripheriebaustein .....	33	Testhilfe für den KIM-1 .....	49
		Tastatur- und Kassettenrecorder-Interface .....	52

Suchprogramm – formuliert in drei Sprachen .....	27	„PRTSTR“ – Einfache Textausgabe in Assembler-Programmen .....	70
Stringmanipulationen – höchst flexibel .....	31	ASCII-Ausgabe per Interrupt .....	71
Der Individualisten-Prozessor .....	39	Amateurfunk-Automat .....	73
SYM druckt 16 Byte pro Zeile .....	41	Mikrocomputer als Contest-Helfer .....	75
PET, der Alleskönner .....	55	Entfernungsberechnung mit QTH-Kennern .....	77
KIM-Klavier .....	66		
8080-Disassembler .....	67		

Die „unterstrichenen“ BASIC-Listings .....	14	Amerika – so weit wie der nächste Briefkasten .....	42
Anwendungsbeispiele für den Mikroprozessor 6502 .....	22	Der AIM-65 im Amateurfunk .....	48
Mikrocomputer-Störstrahlung .....	24	Kassetten-Chaos .....	54
Wo steht etwas über... ..	26	So laufen KIM-Programme auf dem AIM-65 .....	65
The BASIC Handbook .....	32	Programme vom Bildschirm fotografiert .....	66
Unsichtbares mischt mit – Über die Tücken nicht druckender ASCII-Zeichen .....	38	Rechengeschwindigkeit auf dem Prüfstand .....	69
		Computer-Hobbyisten – was sind das für Leute? .....	72

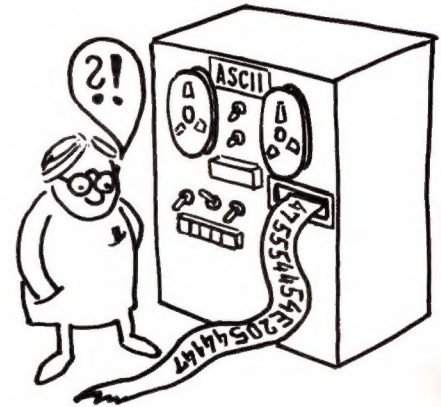


Trotz allen Komforts höherer Programmiersprachen lassen sich viele Software-Probleme nur in der prozessor-spezifischen Maschinensprache lösen, besonders, wenn es um geringen Speicherbedarf oder um hohe Verarbeitungsgeschwindigkeit geht. Am Beispiel des 6502 wird hier gezeigt, wie man zeit-, nerven- und speichersparend programmiert. Dabei wird auch auf Unterschiede zwischen den heute gängigen Mikroprozessoren eingegangen.

Herwig Feichtinger

# Bits und Bytes

Eine kleine Einführung in die Maschinenprogrammierung



## Bits, Bytes, Worte, Zahlen

Grundsätzlich besteht ein Mikrocomputer aus der CPU, dem Mikroprozessor also, einem Programmspeicher, der meist als ROM (read-only memory) ausgelegt ist, einem Datenspeicher (RAM, random access memory) und Bausteinen zur Ein- und Ausgabe von Informationen.

Daten und Programme in RAM und ROM werden vom Mikroprozessor „wortweise“ über den Datenbus transferiert. Bei 8-bit-Prozessoren (SCMP, 8080, 8085, 6800, 6502, Z-80 usw.) ist ein solches Wort acht bit lang, bei 16-bit-Prozessoren (TMS 9900, 8086, 68000, Z-8000) ist es 16 bit lang. Ein Bit ist die kleinstmögliche Informationseinheit, nämlich eine Ja-Nein-Entscheidung. Ein Byte ist eine zusammenhängende Folge von 8 Bits, dies gilt auch im Sprachgebrauch der 16-bit-Prozessoren. Ordnet man den acht Bits die binäre Wertigkeit von  $2^0$  bis  $2^7$  zu, so kann ein Byte die dezimalen Werte 0...255 annehmen.

Bei 8-bit-Prozessoren ist der Datenbus 8 bit breit, d.h. er besteht aus acht Leitungen, die die CPU mit den Speichern verbinden. Bei 16-bit-Prozessoren ist er meist 16 bit breit; manchmal spaltet man das 16-bit-Datenwort aber auch in zwei Bytes auf, die nacheinander auf den Datenbus gegeben werden. Dadurch geht zwar der Geschwindigkeitsvorteil der 16-bit-Prozessoren verloren, man spart aber acht Pins am CPU-Gehäuse (Beispiel: TMS 9980). Den gleichen Effekt erzielt man durch Multiplexen des Datenbus mit dem Adreßbus (8086).

## Hexadezimale Darstellung

Wenn Sie sich Mikrocomputer-Programme oder Speicherinhalte auf dem Display eines kleinen Systems oder auch in Zeitschriften und Büchern ansehen, so werden Sie feststellen, daß die einzelnen Bytes nicht als Folge von acht Nullen und Einsen, also als Bitfolge, wiedergegeben werden, sondern als Ziffern und Buchstaben, nämlich hexadezimal. Dabei steht eine „Ziffer“ (0...F) für vier Bits und repräsentiert eine Dezimalzahl zwischen 0 und 15. Hier eine kleine Tabelle:

Hex	Dezimal	Binär
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Ein Byte oder acht Bits können also durch zwei Hexadezimal-Ziffern dargestellt werden. Da dies dezimal nicht geht, verwenden die meisten Mikrocomputer auch eine hexadezimale Anzeige. Vorsicht: Die Bitfolge 1011 0111 ist zwar hexadezimal B7, aber nicht dezimal 117! Wir wollen uns hier aber

nicht weiter um Zahlenumrechnungen kümmern – diese Problematik wurde in der FUNKSCHAU ja schon mehrfach, z. T. auch mit Taschenrechner-Umwandlungsprogrammen, abgehandelt.

Einige wenige Mikrocomputer, z. B. der H-8 von Heathkit, arbeiten nicht mit hexadezimaler, sondern oktaler Ein- und Ausgabe. Dabei repräsentiert eine Ziffer von 0...7 drei Bits. Pro Byte werden dann drei Ziffern benötigt. Bei den moderneren Mikrocomputern konnte sich das Verfahren der oktalten Darstellung nicht durchsetzen.

## ASCII

Will man in einem Mikrocomputer nicht nur Zahlen, sondern auch Buchstaben, z. B. Text oder Befehle einer höheren Programmiersprache speichern, so wird jeweils ein Byte verwendet, um ein Zeichen zu repräsentieren. Dies geschieht nach dem sog. ISO-7-bit-Code bzw. ASCII (American Standard Code for Information Interchange). Mit 8 bit lassen sich bis zu 256 unterschiedliche Zeichen darstellen, wovon „nur“ 128 für Schriftsymbole und der Rest entweder gar nicht oder für Grafiken genutzt werden.

Die acht Bits des ISO-Codes lassen sich natürlich wieder hexadezimal darstellen. Und so lassen sich die Schriftsymbole der Ziffern 0...9 z. B. einfach dadurch in das hexadezimale ASCII-Äquivalent umrechnen, indem man (hex) 30 addiert. So entspricht der Ziffer 5 z. B. der Hex-Code 35. Die Buchstaben sind alphabetisch mit A beginnend ab hex 41 durchnummeriert. Eine vollständige Tabelle der ASCII-Zeichen



findet sich in FUNKSCHAU 1979, Heft 7, Seite 397. Allerdings braucht man sich darum nur zu kümmern, wenn der Mikrocomputer über alphanumerische Ein- und Ausgabemöglichkeiten verfügt.

Zur Darstellung von Schriftzeichen gibt es auch andere Codes, die in der Mikrocomputer-Technik aber nicht gebräuchlich sind, z. B. der 5-bit-Baudot-Code, der im Fernschreibnetz üblich ist, oder der bei manchen elektrischen Schreibmaschinen verwendete EBCDIC-Code. Eine eventuell notwendige Codeumsetzung ist mit einem Mikrocomputer per Software leicht möglich. So wurde etwa ein Programm zum Umsetzen von ASCII in Baudot in FUNKSCHAU 1979, Heft 1, für den 6502 beschrieben.

Bevor wir uns den Innereien eines Mikroprozessors zuwenden, noch ein Wort über den Ausdruck „Maschinensprache“. Er stammt aus einer Zeit, als die Computer noch aus Tausenden einzelner Transistoren und einiger Mechanik wie Relais und Trommelspeicher bestanden. „Maschine“ war für diese lärm- und hitzeerzeugenden Stromfresser wohl durchaus das richtige Wort. Ein Maschinenprogramm besteht lediglich aus einer Folge von Bytes, der man zunächst nicht viel mehr entnehmen kann als einer mit japanischen Schriftzeichen gedruckten Bedienungsanleitung.

### Register: Speicher in der CPU

Auch im Mikroprozessor selbst gibt es Speicherzellen, die man als „Register“ bezeichnet. An dieser Stelle müssen wir, um nicht in graue Theorie zu verfallen, konkret werden und uns die Register eines realen, handelsüblichen Prozessors ansehen. Wir wählen die CPU 6502, weil sie recht verbreitet und in einer Reihe preiswerter Einplatinen-Computer zu finden ist, wie KIM-1, SYM-1, AIM-65 und PC-100. Dem Leser mag auffallen, daß solche hochwertigen Computer wie der PET 2001 hier nicht genannt sind. Solche Geräte sind für das Arbeiten in BASIC konstruiert; das Programmieren in der Maschinensprache der CPU ist bei ihnen sehr umständlich. Trotzdem sind dafür oft sog. Monitor-Programme lieferbar, z. B. TIM für den PET.

Unsere Wahl des 6502 bedeutet keinesfalls, daß die Besitzer anderer Prozessoren alles folgende überblättern müssen; ganz im Gegenteil werden sie oft „ihre“ Prozessoren im Vergleich mit dem 6502 erwähnt finden, wobei dieser manchmal auch Federn lassen muß.

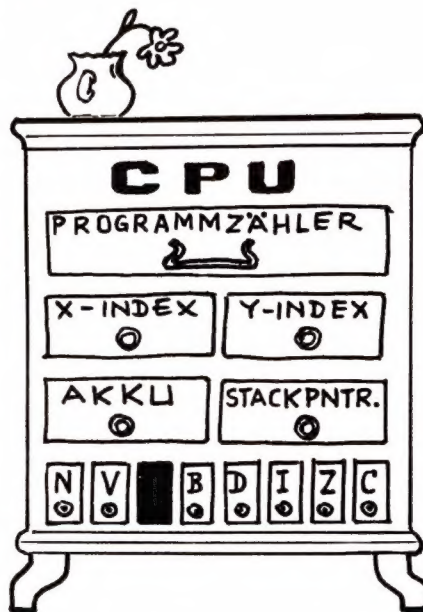
### Der Programmzähler

Doch zurück zu den Registern! Jeder Prozessor besitzt zunächst einmal einen Programmzähler, der ihm sagt, wo der nächste Befehl im Speicher zu finden

ist. Der Programmzähler ist ebenso breit wie der Adreßbus der CPU, bei 8-bit-Prozessoren meist 16 bit (damit lassen sich  $2^{16} = 65536$  Bytes adressieren) und bei 16-bit-Prozessoren im Idealfall 32 bit (z. B. 68000 oder 8086). Manche 16-bit-CPU's besitzen nur einen 14- oder 15-bit-Adreßbus bzw. -Programmzähler (TMS 9980/9900), was dazu führt, daß der adressierbare Speicherbereich sogar kleiner ist als der ihrer 8-bit-Kollegen.

### Der Akku

Das wichtigste Register in der CPU ist der Akku, deshalb besitzt der 6800 gleich zwei davon. In der Regel steht aber nur ein Akku zur Verfügung, und nur in ihm kann man arithmetische Operationen wie Addieren und Subtrahieren ausführen. Der Akku ist bei 8-bit-CPU's 8 bit lang.



Der 16-bit-Mikroprozessor TMS 9900 besitzt wie der 68000 keinen speziellen Akku. Von den meisten anderen CPU's unterscheidet er sich darin, daß sich arithmetische Operationen gleichberechtigt in seinen 16-bit-Registern ausführen lassen, die übrigens nicht innerhalb der CPU liegen, sondern im RAM-Bereich.

Während die meisten Prozessoren mit dem 6502 noch das Vorhandensein eines Akkus gemeinsam haben, hört diese Gemeinsamkeit schon bei den Indexregistern X und Y auf. Der 6800 hat z. B. nur ein einziges Indexregister, jedoch mit 16 bit Länge; andere Prozessoren kennen diese Einrichtung überhaupt nicht.

### X- und Y-Register

Der Mikroprozessor 6502 enthält zwei weitere Register, in die vom Programm Daten hineingeschrieben werden können, nämlich die beiden 8-bit-

Indexregister X und Y. „Index“ bedeutet hier „Adressenversatz“ – eine nützliche Einrichtung, auf die wir bei der Besprechung der Adressierungsarten noch kommen werden. X und Y eignen sich zwar nicht für arithmetische Operationen, lassen sich aber z. B. zum Aufwärts- und Abwärtszählen einsetzen.

### Das Statusregister

Ein weiteres CPU-Register wird von Akku, X- und Y-Register bei der Ausführung bestimmter Befehle beeinflusst: das Statusregister.

Es enthält ebenfalls acht Bits, von denen jedes einen bestimmten Zustand (Status) signalisiert – sozusagen mit einer Flagge, die einzelnen Bits heißen nämlich „Flags“. Sie zeigen an, ob das Ergebnis der letzten Operation Null (Zero) ist, ob ein Übertrag (Carry) z. B. bei einer Addition auftrat, ob das höchstwertige Bit einer Zahl bei der letzten Operation 1 war (Negativ-Flag) und ob ein Software-Interrupt (Break-Befehl) auftrat. Dann gibt es noch etwas beim 6502, was die Besitzer anderer Prozessoren manchmal vor Neid erblasen läßt: das Dezimal-Flag. Ist es 1, also gesetzt, so addiert und subtrahiert der Prozessor dezimal; ist es rückgesetzt, erfolgen diese arithmetischen Operationen binär bzw. hexadezimal. Mit dem Dezimal-Flag kann man also die CPU zwischen zwei Zahlensystemen umschalten.

Dann existiert noch ein Status-Bit, mit dem man externe Interrupts (Programm-Unterbrechungen durch Hardware-Impulse, z. B. Datenanforderung eines Druckers) verhindern kann. Dieses Flag heißt deshalb „Interrupt Disable“.

Das sog. Overflow-Statusbit (V) wird nur bei einigen wenigen Operationen beeinflußt und braucht uns hier noch nicht zu interessieren. Ein letztes Bit im Statusregister ist völlig unbenutzt und liegt in der CPU hardwaremäßig auf log. 1.

Ein weiteres CPU-Register wurde noch nicht erwähnt: der Stackpointer. Auch er ist ein 8-bit-Register (bei anderen Prozessoren z. T. 16 bit) und dient zur Zwischenspeicherung von Adressen und Daten. Doch hierzu später; um dieses Register brauchen wir uns erst bei der Behandlung von Unterprogrammen und Interrupts wieder zu kümmern.

### Ein Befehl: 1...3 Bytes

Je nach Befehl holt sich der Prozessor 1...3 Bytes aus dem Programmspeicher und decodiert sie. (Es gibt auch 8-bit-Prozessoren, die nur 1...2-Byte-Befehle kennen, wie der SCMP, oder bei denen sogar vier Bytes pro Befehl denkbar sind, z. B. der Typ Z-80.)



Das erste Byte des Befehls ist der sog. Operationscode. Die Operationscodes des 6502 sind in FUNKSCHAU 1979, Heft 11, S. 657, abgedruckt. Sie stellen den eigentlichen Befehl dar.

Manche Befehle führen eine Operation aus, für die kein Argument benötigt wird; z. B. löst der Befehl SED (Set Decimal Flag, hex F8) eine CPU-interne Operation aus, für die keine Daten benötigt werden. Andere Befehle benötigen zur Ausführung zusätzliche Daten. So wäre etwa der Befehl LDA (Load Accu) sinnlos, wenn man nicht angeben würde, womit der Akku geladen werden soll. Will man den Akku mit dem Inhalt einer beliebigen Speicherzelle laden, so muß dem Befehl LDA eine 16-bit-Adresse folgen, die als vierstellige Hexadezimal-Zahl geschrieben werden kann, was zwei Bytes entspricht. Der Befehl setzt sich also hier aus drei Bytes zusammen – einem Byte für den Operationscode und zwei für das Argument.

Eine weitere Möglichkeit ist, daß der Akku direkt mit irgendwelchen Daten geladen werden soll, z. B. mit 56. Der Befehl lautet hier A9 56; der Operationscode ist A9, und 56 das Argument. Hier werden nur zwei Bytes benötigt.

Die CPU erkennt am Operationscode bereits, wie viele Bytes noch zu dem Befehl gehören, und stellt den Programmzähler während der Befehlsausführung entsprechend weiter.

Sieht man sich im Speicher eines Mikrocomputers um, so erkennt man an den hexadezimalen Operationscodes natürlich nicht gleich, wie viele Datenbytes nach ihnen folgen bzw. wo der nächste Befehl beginnt. Hier leistet ein „Disassembler“ gute Dienste: Er übersetzt nicht nur die Hex-Codes in die mnemonische Form (z. B. A9 in LDA), sondern erkennt auch die richtige Befehlslänge. Disassembler sind selbst Programme; für den 6502 wurde ein solches Programm in FUNKSCHAU 1978, Heft 21 veröffentlicht. Der Mikrocomputer AIM-65 hat bereits einen Disassembler im ROM „eingebaut“. Voraussetzung für die Anwendung eines Disassemblers ist allerdings grundsätzlich eine alphanumerische Ausgabe-möglichkeit.

## Befehle und Daten

Eine naheliegende Frage ist nun: Da sowohl die Operationscodes als auch die zu verarbeitenden Daten, die ihnen folgen, zunächst einmal nur binäre Zahlen sind, woher weiß dann die CPU, was Befehle und was Daten sind?

Die Antwort ist sehr einfach: Sie weiß es nämlich nicht! Der Programmierer hat dafür zu sorgen, daß er das Programm an einer Adresse startet, an der auch tatsächlich ein Operationscode

steht. Dann ist alles gerettet: Ab sofort weiß ja die CPU, wie viele Daten-Bytes dem Operationscode folgen, d.h. wo sie den nächsten Befehlscode findet.

Kritisch wird es erst bei Sprungbefehlen. Wenn der Programmierer nicht aufgepaßt hat, springt das Programm u.U. einmal nicht auf einen Operationscode, sondern auf die ihm folgenden Daten und interpretiert diese als Befehl. Dies kann katastrophale Folgen haben: Steht das Programm im RAM, so kann es sich u.U. selbst zerstören, indem undefiniert Daten an irgendwelche Adressen gespeichert werden. In solchen Fällen muß man oft das gesamte Programm neu eingeben.

## Ein erstes Programm

Wir könnten nun schon versuchen, ein einfaches Programm selbst zu schreiben. Hier gehen wir aber einmal den umgekehrten Weg: Wir analysieren einen vorhandenen Speicherinhalt, der folgendermaßen aussieht:

```
0200 A9 05 18 69 07 8D 10 02
0208 4C 4F 1C (beim KIM-1)
0208 4C BF E0 (beim AIM-65)
```

Selbstverständlich stehen hier nicht alle Bytes an nur zwei Adressen, sondern A9 steht bei 0200, 05 bei 0201 usw. An der Adresse 0208 steht hier ein Rücksprungbefehl zum Monitor-Programm des jeweils verwendeten Mikrocomputers, der mit dem eigentlichen Programm nichts zu tun hat, sondern lediglich ermöglicht, daß nach dem Programmablauf wieder die Eingabe und Anzeige von Befehlen und Daten über Tastatur und Display möglich sind.

Die hier aufgelisteten Bytes können bei den verschiedenen Mikroprozessoren (6800, Z-80 usw.) eine ganz unterschiedliche Bedeutung haben. Nicht zuletzt deshalb eignen sich diese hexadezimalen Codes für das Dokumentieren von Programmen nicht besonders gut – der Besitzer eines anderen Prozessor-typs kann mit ihnen nichts anfangen.

Zum Analysieren des Speicherinhaltes können wir uns entweder einer Operationscode-Tabelle bedienen, wir können aber auch einen Disassembler benutzen, wie er bereits im AIM-65-Monitorprogramm vorhanden ist. Er liefert uns folgenden Ausdruck:

```
0200 A9   LDA # 05
0202 18   CLC
0203 69   ADC # 07
0205 8D   STA 0210
0208 4C   JMP E0BF
```

Ganz links steht dabei jeweils die Adresse des Operationscodes, dann folgt der Operationscode und der Be-

fehl in mnemonischer Form (drei Buchstaben) nebst dem Argument, sofern eines vorhanden ist.

Andere Disassembler lassen entweder den hexadezimalen Operationscode weg, oder aber sie drucken die Befehle etwa in folgender Form:

```
0200 A9 05   LDA # 05
```

Was tut nun dieser erste Befehl? LDA bedeutet „Load Accu“, und das Doppelkreuz vor dem Argument besagt, daß der Akku nicht mit dem Inhalt einer Adresse außerhalb des Programms, sondern direkt mit dem Wert des dem Operationscode folgenden Byte geladen werden soll. Dieses direkte Laden wird in der Operationscode-Tabelle mit der „Adressierungsart Immediate“ bezeichnet.

Der nun folgende Befehl CLC löscht das Übertrags-Flag im Status-Register. Dies ist notwendig, weil der nachfolgende Additionsbefehl, der zum Akkuinhalt 7 addieren soll, das Übertrags-Flag (Carry) mit einbezieht, um Additionen mehrstelliger Zahlen zu erlauben. Der Zustand des Carry-Flag ist vor der Ausführung des Programms nicht definiert; es kann zufällig irgendeinen Zustand angenommen haben.

Nach dem ADC-Befehl (Adressierungsart wieder „Immediate“) steht im Akku die hexadezimale Summe von 5 und 7. (Das Monitorprogramm des Mikrocomputers hat vor der Programmausführung automatisch dafür gesorgt, daß der Prozessor im Hexadezimal-Modus arbeitet, d.h. das Dezimal-Flag gelöscht ist.)

Da wir nicht direkt in den Akku hineinsehen können, muß das Ergebnis an irgendeine Speicherzelle transferiert werden, an der wir es später prüfen können. Dafür sorgt der Befehl STA 0210; er speichert den Akkuinhalt an die Adresse 0210. Bitte beachten Sie: Das hexadezimale Programm ist so aufgebaut, daß nach dem Operationscode 8D zunächst das niederwertige, dann das höherwertige Byte der Zieladresse folgt; dies ist bei den meisten Mikroprozessoren üblich, nur der 6800 von Motorola macht hier eine Ausnahme.

Der letzte Befehl, JMP, ist systemabhängig und führt zu einem Sprung in das Monitorprogramm des Mikrocomputers, beim AIM-65 an die Adresse E0BF. Ebenso wie beim vorhergegangenen STA-Befehl ist die Adressierungsart „absolut“: Es wird nach dem Operationscode eine 16-bit-Adresse als Argument genannt.

Wenn wir unseren Mikrocomputer mit dem Programm laden und es an der Adresse 0200 starten, muß danach an der Adresse 0210 das Additionsergebnis 0C (dezimal 12) stehen.



## Zero-Page-Adressierung

Was ist eine „Page“?

Wie schon erwähnt, wird der adressierbare Speicherbereich eines Mikrocomputers von 16 Bits charakterisiert. Diese 16 Bits lassen sich mit vier hexadezimalen „Ziffern“ (0...F) darstellen. Der gesamte Adressenbereich reicht also von 0000 bis FFFF.

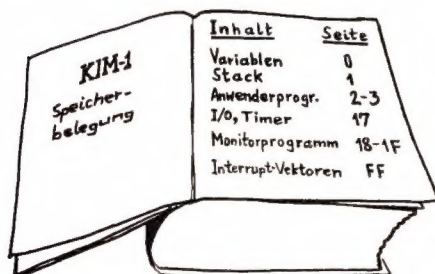
Eine Adresse kann als Folge zweier 8-bit-Bytes betrachtet werden; dies geschieht ja auch bei der gerade erwähnten „absoluten“ Adressierung. Während sich mit zwei Byte der gesamte Adressenbereich von 65 536 Byte überstreichen läßt, kann man mit dem niederwertigen der beiden Adreßbytes nur  $2^8 = 256$  Byte adressieren. Und einen solchen Bereich von 256 Byte nennt man auch „Page“, zu deutsch „Seite“. Der gesamte Adressenbereich eines Mikrocomputers kann also durch 256 Seiten mit je 256 Byte dargestellt werden.

Der untere Adressenbereich (Page 0, 1, 2,...) ist in 6502-Systemen meist mit RAM, der obere mit dem das Monitorprogramm enthaltenden ROM belegt. Um Hardware zu sparen, werden in vielen Einplatinen-Computern nicht alle 16 Adreßleitungen decodiert, so daß sich manche Adressen „spiegeln“; so findet man etwa beim KIM-1 in Page 22 die gleichen Daten wie in Page 2, d.h. die Daten in den Zellen 0200...02FF sind scheinbar identisch mit denen im Bereich 2200...22FF.

### Die nullte Seite

Diejenige „Page“, bei der das höherwertige Adreßbyte 00 ist, hat beim 6502 eine besondere Bedeutung. Diese „Zero Page“ – sie ist eine eigene Adressierungsart in der Operationscode-Tabelle – erlaubt Speicheroperationen mit nur zwei Bytes pro Befehl, was ideal für die Variablen eines Programms ist.

Ein Beispiel: Bei unserem Additionsprogramm wurde das Ergebnis im Akku mit einem Drei-Byte-Befehl (8D 10 02) an die Adresse 0210 gespeichert. Wenn wir es stattdessen bei 0010 abspeichern wollen, könnten wir natürlich den Befehl auf 8D 10 00 ändern; wir können aber auch die Adressierungsart „Zero Page“ verwenden. Der Operationscode



Inhalt	Seite
Variablen	0
Stack	1
Anwenderprogr.	2-3
I/O, Timer	47
Monitorprogramm	18-1F
Interrupt-Vektoren	FF

ist dann laut Tabelle 85, und der gesamte Befehl hat nur noch zwei Bytes: 85 10. Disassembliert sähe das z. B. so aus:

0205 85 STA 10

Die nächste freie Adresse ist 0207, hier könnte jetzt – also ein Byte früher als vorher – der JMP-Befehl stehen. Wollen wir ihn nicht verschieben, so müssen wir den nun freien Platz auffüllen, nämlich mit dem Befehl „No Operation“:

0207 EA NOP

Er besteht aus nur einem Byte, da er kein Argument benötigt, und verändert nichts in der CPU außer dem Programmzähler; d.h. das Statusregister bleibt unbeeinflusst.

### Noch einmal: Das Statusregister

Sehen wir uns doch einmal an, was beim Ablauf des einfachen Additionsprogrammes mit dem Statusregister in der CPU geschieht. Hier eine zusammenfassende Aufstellung:

0200

Das Null-Flag (Zero-Flag) wird auf Null rückgesetzt, weil der Akku mit einem Wert ungleich Null geladen wird.

0202

Das Carry-Flag wird gelöscht.

0203

Es ändert sich nichts: Da kein Übertrag bei der Addition auftritt, bleibt das Carry-Flag gelöscht. Auch das Zero-Flag bleibt Null, da das Ergebnis ungleich Null ist.

0205

Operationen, die den Inhalt eines CPU-Registers in den RAM-Speicher transferieren, ändern grundsätzlich kein Status-Bit; es ändert sich also wieder nichts.

Welches Bit im Statusregister bei welchen Befehlen wie beeinflusst wird, ist wiederum von Prozessortyp zu Prozessortyp unterschiedlich. Es ist daher am besten, in Zweifelsfällen im Programmierhandbuch des CPU-Herstellers nachzusehen.

### Indizierte Adressierung

Die beiden Index-Register des 6502 blieben bisher unbenutzt. Sie lassen sich jedoch recht nützlich einsetzen, wie das folgende Programmbeispiel zeigt, dessen Aufgabe es ist, den Speicherbereich 0301...037F voller Nullen zu schreiben. (Der Mikroprozessor 6800 gestattet es, den Inhalt beliebiger Adressen ohne den Umweg über den Akku auf Null zu setzen [CLR]. Beim 6502 ist dies leider nicht möglich.)

```
0200 A9 00 LDA # 00
0202 A2 7F LDX # 7F
0204 9D 00 03 STA 0300,X
0207 CA DEX
0208 D0 FA BNE 0204
0204 4C ... JMP Monitor
```

Das X-Register dient hier als Adressenversatz für den STA-Befehl. Die tatsächliche Adresse, an die der Akkuinhalt gespeichert wird, ergibt sich aus der Summe der dem Operationscode 9D folgenden beiden Adreßbytes und dem Inhalt des X-Registers. Nach dem Programmstart wird zunächst der Akku mit 00 und das X-Register mit 7F geladen. Dann wird der Akku an die Adresse (0300 + X) gespeichert, was zunächst 037F ergibt. Schließlich wird das X-Register um eins erniedrigt (dekrementiert), und das Spiel beginnt von neuem.

Auch der DEX-Befehl beeinflusst das Status-Register. Und deshalb läuft die Programmschleife nur solange, bis X Null geworden ist. Dafür sorgt der Befehl BNE, sprich „Branch on Not Equal“. Seltsamerweise hat der Disassembler hier eine Zwei-Byte-Adresse hinter den mnemonischen Befehl BNE gesetzt, obwohl es sich um einen Zwei-Byte-Befehl mit nur einem Byte als Argument handelt. Wieso dieses?

### Relative Adressierung

Der 6502 kennt zwei Sorten von Sprungbefehlen: bedingte und unbedingte. Erstere heißen „Branch“ (Verzweigung), letztere einfach JMP für Jump.

Während bei JMP stets eine Zwei-Byte-Adresse folgt, steht hinter den Branch-Befehlen nur ein Byte, das einen Adressenversatz angibt, der positiv (00...7F) oder negativ (80...FF) sein kann. 00 entspricht der dem Branch-Befehl unmittelbar folgenden Adresse, bei 02 würden die folgenden zwei Bytes übersprungen usw. Etwas komplizierter als bei Vorwärtssprüngen ist es, wenn der Branch-Befehl nach „rückwärts“ verzweigt. Dabei gibt es eine einfache Methode, sich große Rechnereien zu ersparen: Man braucht nur (hexadezimal) abzuzählen, wo man hinspringen will. Will man z. B. wie bei unserem letzten Programmbeispiel sechs Byte zurückspringen, so braucht man nur, beginnend mit der dem Branch-Befehl folgenden Adresse 020A, bis 0204 zurückzuzählen: 00, FF, FE, FD, FC, FB, FA – und schon sind wir bei 0204, der Zieladresse des bedingten Sprunges. Schon nach kurzer Übung kann man das hexadezimale Vor- und Rückwärtszählen in- und auswendig.

Auch hier ist wieder größte Vorsicht geboten, um sicherzustellen, daß der Sprung auch tatsächlich zum gewünschten Operationscode führt; springt das Programm fälschlich auf irgendwelche Daten, so läuft es u.U. „Harakiri“.

Branch-Befehle verzweigen nur dann, wenn bestimmte Flags im Statusregister gesetzt oder gelöscht sind. In unserem Beispiel wird der BNE-Sprung



nur dann ausgeführt, wenn das Ergebnis der letzten Operation (DEX) ungleich Null war, wenn also das Zero-Flag in der CPU gelöscht war. Der Befehl BEQ verzweigt dagegen bei gesetztem Zero-Flag. Andere Branch-Befehle verzweigen abhängig von den Carry-, Overflow- oder Negativ-Flags. Manche Prozessoren – der 6502 leider nicht – verfügen auch über die Möglichkeit, abhängig von einem „Parity-Bit“ zu verzweigen, das eine Prüfsumme des Akkuinhaltes darstellt.

Der 6502 kennt auch keinen unbedingten Sprung mit relativer Adressierung. Die Abhilfe ist hier allerdings einfach. Sie besteht aus der Befehlsfolge CLC (Clear Carry) und BCC (Branch on Carry Clear) oder Gleichwertigem, z. B. SEC/BCS. Beim 6800 ist dieser Trick nicht erforderlich.

## Das Monitorprogramm

An dieser Stelle muß auf etwas eingegangen werden, das bisher als selbstverständlich angesehen wurde, aber doch einige Bemerkungen wert ist: das Monitorprogramm.

Wie schon einmal erwähnt, steht in einem Mikrocomputer dieses Programm im ROM, damit es auch nach dem vorübergehenden Abschalten der Versorgungsspannung wieder zur Verfügung steht. Nun gibt es aber doch Systeme, die gar kein ROM haben – wie kommt dann das Programm hinein? Ein solches System ist z. B. das von Horst Pelka 1977 in der FUNKSCHAU beschriebene 8080-System: Hier wurden einfach die Daten über den Datenbus bei gleichzeitigem Einstellen der Adresse mit Schaltern ohne Hilfe der CPU direkt vom Programmierer in das System-RAM eingegeben.

Die Methode, ein Programm binär mit Schaltern für Adressen und Daten einzugeben, ist zwar vielleicht lehrreich, aber zeitraubend, fehlerträchtig und oft entmutigend. Käufliche Einplatinen-Computer (KIM-1, AIM-65 usw.) besitzen daher ein ROM, das ein fest gespeichertes Programm enthält. Dieses sog. „Monitorprogramm“ fragt z. B. ein Hexadezimal-Tastenfeld ab (beim AIM-65 sogar ein alphanumerisches), decodiert die Tasten und speichert die eingegebenen Daten an die gewünschten Adressen ab. Außerdem ermöglicht das Monitorprogramm (von Billigcomputern wie dem SCMP/MK-14 einmal abgesehen) auch das Abspeichern von fertig entwickelten Programmen auf eine normale Tonband-Kassette, damit man längere Programme nach dem Einschalten des Systems nicht jedesmal wieder neu eingeben muß. Meist wird auch die Anzeige von Adressen und Daten auf einem Siebensegment-Display oder einer alphanumerischen Anzeige mit

Hilfe des Monitorprogramms realisiert; es kann z. B. per Software hexadezimale Zahlen in den Siebensegment-Code umwandeln.

Monitorprogramme können – je nachdem, wie komfortabel sie sind – etwa zwischen 1 KByte und 8 KByte im System-ROM einnehmen. Gewisse Teile des Monitorprogrammes, nämlich die Unterprogramme, können von einem Programm mitbenutzt werden, die der Anwender in das System-RAM geschrieben hat. Zu beachten ist schließlich noch, daß das Monitorprogramm einige wenige Adressen im RAM als Zwischenspeicher mitbenutzt, in denen keine Anwenderprogramme stehen dürfen, da diese sonst überschrieben werden.

## Der „Kellerspeicher“

### Unterprogramme

Wie in den höheren Programmiersprachen, so gibt es auch in der Maschinensprache die Möglichkeit, Unterprogramme zu verwenden. Was in BASIC als Befehl GOSUB heißt, wird beim 6502 mit JSR (Hex-Code 20) bezeichnet, und der BASIC-Befehl RETURN hat sein 6502-Äquivalent in RTS (Return from Subroutine, hex 60).

Unterprogramme sind immer dann sinnvoll, wenn eine gleichartige Befehlsfolge mehrmals im Programm gebraucht wird, z. B. um die drei Speicherzellen 0000, 0001 und 0002 mit Nullen zu füllen. Das Unterprogramm sähe dabei so aus:

0250 A9 00	LDA # 00
0252 85 00	STA 00
0254 85 01	STA 01
0256 85 02	STA 02
0258 60	RTS

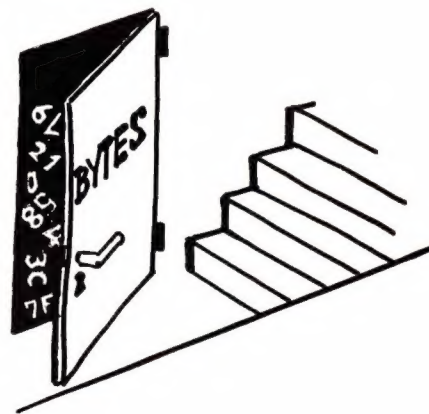
Dem geneigten Leser wird auffallen, daß zum Abspeichern des Akkuinhaltes (00) die Zero-Page-Adressierung für die Adressen 0000...0002 verwendet wird. Am Ende des Programms steht der Rücksprung-Befehl RTS. Vorsicht: Starten Sie das Programm nicht an der Adresse 0250 ohne JSR-Aufruf! Es weiß sonst bei RTS nicht mehr, wohin es springen soll und läuft wiederum mit größter Wahrscheinlichkeit „Harakiri“.

Ruft man dieses Unterprogramm auf, so geschieht das einfach mit JSR 0250, hexadezimal 20 50 02. Dies kann innerhalb des Hauptprogrammes beliebig oft geschehen. Auch kann ein Unterprogramm von einem „übergeordneten“ Unterprogramm aufgerufen werden. Woher weiß es nun aber, wohin es zurückspringen muß, wenn die Subroutine-Befehlsfolge durchlaufen ist? Schließlich soll ja dann der dem JSR-Befehl folgende Befehl ausgeführt werden, d.h. das Unterprogramm soll zum

nächsten Befehl des Hauptprogrammes zurückspringen. Irgendwie muß sich das Unterprogramm also die Rücksprungadresse „merken“.

### Der Stack als Adressenspeicher

Die CPU speichert die Rücksprungadresse bei Unterprogrammaufrufen schon während des JSR-Befehls in einen bestimmten Speicherbereich, den man „Stack“ nennt; in der deutschen Literatur findet man ab und zu auch das abschreckende Wort „Kellerspeicher“.



Beim 6502 liegt der Stack durch die CPU-interne Hardware fest im Bereich 0100...01FF, also in Page 1. Damit die CPU weiß, wohin sie mit der Adresse soll, die beim JSR-Befehl gespeichert werden muß, besitzt sie ein Register namens „Stackpointer“, zu deutsch Kellerspeicherzeiger, das acht Bits umfaßt und vom Monitorprogramm des Mikrocomputers – solange kein Unterprogramm aufgerufen wird – auf FF gesetzt wird. Der Stackpointer zeigt also zunächst auf die Adresse 01FF.

Nehmen wir an, der Unterprogramm-Aufruf stünde an der Adresse 0220:

```
0220 20 50 02 JSR 0250
```

An der Adresse 0223 stünde dann der nächste Befehl im Programm, wo nach der Abarbeitung des Unterprogramms fortgefahren werden soll. Beim JSR-Befehl geschieht nun folgendes:

Die Adresse des letzten Bytes beim JSR-Befehl (hier 0222) wird in zwei Hälften gespalten. Der höherwertige Adressenteil (also 1 Byte, hier 02) wird an die Adresse gespeichert, auf die der Stackpointer zeigt (hier 01FF). Dann wird der Stackpointer um 1 erniedrigt, und der niederwertige Adressenteil (hier 22) wird an die Adresse 01FE gespeichert. Schließlich wird der Stackpointer nochmals dekrementiert, so daß er jetzt auf 01FD zeigt, und es erfolgt ein Sprung zum Unterprogramm.



Wenn dieses abgearbeitet ist, erfolgt bei RTS ein Rücksprung, indem der Stackpointer wieder inkrementiert wird; zuerst wird nun das niederwertige Byte und dann – nach nochmaliger Inkrementierung des Stackpointers – das höherwertige Byte der Rücksprungadresse in den Programmzähler der CPU transferiert. Der Stackpointer zeigt nun wieder auf 01FF. Bei verschachtelten Unterprogrammen, wenn also ein Unterprogramm ein oder mehrere weitere aufruft, müssen nacheinander mehrere Rücksprungadressen gespeichert werden, so daß der Stackpointer immer „tiefer“ dekrementiert wird. Darauf ist zu achten, wenn man ein Programm in Page 1 stehen hat, das dann u.U. von den zu speichernden Rücksprungadressen überschrieben werden kann.

Da die Rücksprungadresse nicht die Adresse des nächsten Befehls im Hauptprogramm, sondern die des letzten Byte im JSR-Befehl ist, sorgt die CPU automatisch noch dafür, daß der Programmzähler bei RTS zusätzlich noch um 1 erhöht wird, so daß er auf den folgenden Befehl zeigt.

### So rettet man den Akku auf den Stack

In unserem Unterprogramm-Beispiel, das drei Speicherzellen mit Null beaufschlagte, war natürlich nach dem Unterprogramm-Aufruf der Akkuinhalt gelöscht. Zuweilen stehen aber im Akku wichtige Dinge, die man auch nach einer längeren Befehlsfolge (die kein Unterprogramm zu sein braucht) wieder weiterverarbeiten möchte. Dann kann man sich der Befehle PHA und PLA bedienen. Das Beispiel macht sofort deutlich, wie das funktioniert.

```
PHA  Akku auf den Stack retten
...  Unterprogramm (zerstört den
      Akkuinhalt)
PLA  Akku vom Stack zurückholen
RTS
```

Der Befehl PHA speichert den Akkuinhalt an die durch den Stackpointer spezifizierte Adresse und erniedrigt dann diesen um 1. PLA dagegen inkrementiert den Stackpointer und lädt den Akku mit seinem ursprünglichen Inhalt.

Leider ist es nicht möglich, X- und Y-Register direkt auf den Stack zu „retten“ – dies geht nur über den Umweg des Akkus. In dem obigen Beispiel kann es – wenn das Unterprogramm auch X und Y ändert – sinnvoll sein, die Befehlsfolge so zu erweitern:

```
PHA  Akku retten
TXA  X in den Akku transferieren...
PHA  ... und retten
TYA  ebenso Y
PHA
...  Unterprogramm
```

```
PLA  Y vom Stack holen
TAY
PLA  X vom Stack holen
TAX
PLA  Akku rückspeichern
RTS
```

Im Gegensatz zum 6502 erlauben es manche andere Mikroprozessoren, auch die übrigen Register ohne Umweg über den Akku auf den Stack zu retten, u.U. sogar automatisch beim JSR-Befehl.

Wieder andere Prozessoren besitzen gar keinen Stack; entweder, weil es bei ihnen keinen Unterprogramm-Befehl gibt (SCMP), oder, weil die Rücksprungadressen bzw. Registerdaten auf andere Weise gespeichert werden (TMS-9900-Familie).

### Der Stackpointer

In den vorangegangenen Beispielen wurde deutlich, daß bei Unterprogrammaufrufen und PHA-Befehlen Daten und Adressen immer an die durch den Stackpointer spezifizierte Adresse abgespeichert werden. Da nach dem Einschalten der Versorgungsspannung der Stackpointer irgendwohin zeigen kann, wird er meist vom Monitorprogramm zunächst auf seinen Normalwert gesetzt, nämlich auf FF:

```
LDX  # FF
TXS
```

Die gleiche Befehlsfolge wird notwendig, wenn man aus einem Unterprogramm nicht über RTS, sondern – z. B. wegen einer Programmverzweigung – über einen JMP-Befehl in das Hauptprogramm oder auch in das Monitorprogramm springen will. Der Stackpointer steht nun ja nicht mehr in seiner „Ruhestellung“ FF und muß neu gesetzt werden. Dies ist besonders dann wichtig, wenn sich im Bereich ab 0100 (Stack-Page) Programme und Daten befinden, die sonst von dem immer tiefer rückenden Stackpointer überschrieben werden könnten.

Allgemein muß der Stackpointer immer dann korrigiert werden, wenn ein JSR-Befehl nicht durch RTS wieder aufgehoben wird. (Gleiches gilt für die noch zu besprechenden „Interrupts“.) Vergißt man das, so wird u.U. nach und nach der ganze durch den Stackpointer adressierbare Speicherbereich überschrieben – und manche Prozessoren haben einen 16-bit-Stackpointer...

### Ein- und Ausgabe über Tastatur und Display

Es gibt bei den heute üblichen Mikrocomputer-Systemen zwei Arten der Ein- und Ausgabe: Einmal über die auf der Platine vorhandene Tastatur und Anzeige bzw. über ein externes Terminal, oder aber über Eingabe-Ausgabe-Leitungen (I/O-Ports). Erstere erlaubt

die Kommunikation mit dem Benutzer des Systems, zweitere dient zur Steuerung und Abfrage externer Geräte, die Meßergebnisse liefern oder die der Mikrocomputer automatisch steuern soll.

### Display-Ansteuerung

Für die Abfrage des Tastenfeldes und die Anzeige auf dem Display oder Terminal besitzt das Monitorprogramm des Mikrocomputers praktisch immer geeignete Unterprogramme, die auch vom Anwenderprogramm her aufgerufen werden können. Sie sind aus dem Monitorprogramm-Listing im Systemhandbuch ersichtlich. Für die Computer KIM-1 und SYM-1 wurden die wichtigsten in FUNKSCHAU 1979, Heft 11, Seite 653, veröffentlicht (in diesem Heft findet sich auch die 6502-Operationscode-Tabelle). Beim AIM-65 sind die Monitor-Unterprogramme ausführlich im Handbuch kommentiert.

Als typisches Beispiel eines Einplatinen-Computers mit Siebensegment-Display und Hexadezimal-Tastatur soll hier der KIM-1 dienen. Er besitzt eine Anzeigeroutine im Monitorprogramm an der Adresse 1F1F. Sie stellt den Inhalt der Zero-Page-Zellen 00FB, 00FA, 00F9 auf dem sechsstelligen Display drei Millisekunden lang dar. Vor und nach ihrem Aufruf ist das Display dunkel. Will man eine ständige Anzeige erreichen, so muß das Anzeige-Unterprogramm in einer Schleife dauernd durchlaufen werden. Will man z. B. auf dem Display lauter Nullen anzeigen, könnte das Programm etwa so aussehen:

```
0200 A9 00  LDA # 00
0202 85 FB  STA FB
0204 85 FA  STA FA
0206 85 F9  STA F9
0208 20 1F 1F JSR 1F1F
020B 4C 08 02 JMP 0208
```

Hier werden zunächst die drei Anzeigebuffer-Zellen gelöscht. Dann folgt eine Programmschleife, bestehend aus dem Unterprogramm-Aufruf zur Anzeige und einem Sprungbefehl. Hat man das Programm an der Adresse 0200 einmal gestartet, kommt man z. B. durch Drücken der Reset-Taste wieder heraus. Eine andere Möglichkeit, das Programm zu unterbrechen, ergibt sich aus der Fähigkeit der Anzeige-Routine, zu erkennen, ob irgendeine Taste auf dem KIM-1 gedrückt ist. Ist das nämlich nicht der Fall, so ist der Inhalt des Akkus bei der Rückkehr aus dem Unterprogramm Null. War eine Taste gedrückt, ist der Akkuinhalt ungleich Null, und das Zero-Flag im Statusregister wird rückgesetzt. Wollen wir dafür sorgen, daß durch Drücken einer Taste aus der Anzeige-Programmschleife zum KIM-Monitorprogramm gesprungen werden kann, so brauchen wir nur folgendes zu ändern:



```
020B F0 FB    BEQ 0208
020D 4C 4F 1C JMP 1C4F
```

Leider ist der „Erfolg“ beim Start dieses Programms deprimierend: Es erfolgt nämlich sofort ein Rücksprung zum Monitorprogramm an die Adresse 1C4F, so daß lediglich der Inhalt der Adresse 0000 auf dem Display und nicht etwa sechs Nullen erscheinen.

Warum? Nun, wir starten das Programm ja mit der Taste GO. Und diese Taste ist höchstwahrscheinlich auch noch nach der 3 ms dauernden Anzeigeroutine gedrückt. Da aber ja ein Rücksprung erfolgen soll, wenn eine Taste gedrückt ist, kommt die Programmschleife zur Anzeige von sechs Nullen nicht zustande.

Was nun? Wir müssen noch eine zweite Schleife einbauen, die dafür sorgt, daß die eigentliche Anzeigeschleife erst dann erreicht wird, wenn die Taste GO wieder losgelassen wird.

```
020B D0 FB    BNE 0208
020D 20 1F 1F JSR 1F1F
0210 F0 FB    BEQ 020D
0212 4C 4F 1C JMP 1C4F
```

Tatsächlich funktioniert es jetzt! Der Unterprogramm-Aufruf bei 0208 dient hier eigentlich nicht der Anzeige, sondern nur der Tastenabfrage. Ist noch eine Taste gedrückt, kann das Programm nicht zur Adresse 020D weiterücken.

Andere Mikrocomputer haben eine solche Überprüfung, ob die letzte Taste noch gedrückt ist, schon innerhalb des Monitorprogramms „eingebaut“. Außerdem muß z. B. beim AIM-65 das Display-Unterprogramm nicht in einer Schleife laufen, weil die Ansteuerung der einzelnen Display-Stellen nicht per Software geschieht. Vielmehr braucht dem AIM-Display nur per Unterprogramm ein neues Zeichen übergeben zu werden, das dann von rechts in das Display geschoben wird. In fast allen Fällen wird, wenn ein Zeichen auf ein Terminal oder einen ASCII-Fernschreiber ausgegeben werden soll, dieses Zeichen im Akku übergeben. Beim AIM-65 gilt dies auch für das 20stellige alphanumerische Display.

#### Tastatur-Abfrage

Der Computer KIM-1 besitzt an der Adresse 1F6A ein Monitor-Unterprogramm namens „GETKEY“. Es fragt das hexadezimale Tastenfeld ab und kehrt mit dem hexadezimalen Wert der Taste im Akku zurück. Aber: Auch hier wird nicht abgefragt, ob die zuletzt gedrückte Taste immer noch niedergedrückt ist. An folgendem kleinen Programm wird das sofort deutlich:

```
0200 20 6A 1F JSR 1F6A
0203 85 F9    STA F9
0205 20 1F 1F JSR 1F1F
0208 4C 00 02 JMP 0200
```

Dieses Programm stellt den Wert einer gerade gedrückten Taste in den niederwertigsten beiden Display-Stellen in Form eines Byte dar. Ist keine Taste gedrückt, so wird 15 angezeigt. Es wird sofort deutlich, daß sich der Mikrocomputer den Wert einer Taste nicht „merkt“, sondern sofort wieder vergißt, wenn sie losgelassen wird. Will man z. B. ein Programm schreiben, um von der Tastatur her mehrere Ziffern wie bei einem Taschenrechner in das Display zu schreiben, so wäre auch hier eine zusätzliche Abfrage nötig, ob die letzte Taste noch gedrückt ist.

Beim AIM-65 ist dies alles wiederum nicht nötig. Das Unterprogramm zur Tastatur-Abfrage wartet in einer Schleife automatisch so lange, bis die vorher gedrückte Taste losgelassen und eine neue Taste gedrückt wird. Um hier von der Tastatur aus auf dem Display zu schreiben, genügt beim AIM folgendes Programm:

```
0200 20 73 E9 JSR E973
0203 4C 00 02 JMP 0200
```

Die Routine an der Adresse E973 sorgt nicht nur für die Tastaturabfrage, sondern stellt die gedrückten Tasten auch gleichzeitig auf dem Display dar.

#### Ein- und Ausgabe per „Memory Map“

Ebenso wie der 6800 besitzt auch der Mikroprozessor 6502 keine besonderen Ein- und Ausgabe-Befehle. I/O-Ports werden vielmehr als gewöhnliche Speicheradressen betrachtet, und alle CPU-Befehle, die eine Speicheroperation beinhalten, können auf I/O-Ports angewandt werden. Der englische Ausdruck für Ports, die normale Speicheradressen sind, ist „memory-mapped I/O“, etwa „Speicherlandkarten-Ein- und Ausgabe“ (schrecklich!).

Eine Besonderheit ist auch, daß die in den Bausteinen 6520, 6522, 6530 oder 6532 integrierten I/O-Ports entweder als Eingang oder Ausgang deklariert werden können. Zu diesem Zweck sind jedem 8-bit-Port zwei 8-bit-Register, also zwei Speicheradressen zugeordnet, beim KIM u. a.:

```
1700 PA (Port A)
1701 PAD (Port-A-Richtung, „Direction“)
```

Nehmen wir an, wir wollten eine Leuchtdiode mit Hilfe eines Schalters ein- und ausschalten. Damit der Mikrocomputer etwas zu tun hat, soll das nicht direkt, sondern über zwei I/O-An-

schlüsse geschehen. Den Schalter schließen wir am Pin PA 7 und die Leuchtdiode am Pin PA 0 an (das ist ganz willkürlich gewählt).

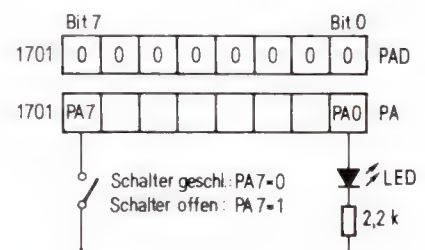
PA 7 muß also als Eingang, PA 0 als Ausgang geschaltet werden. Das geschieht durch Setzen entsprechender Bits im Datenrichtungs-Register PAD; eine Null bedeutet „Eingang“, eine Eins „Ausgang“. (Beim Drücken der Reset-Taste werden alle Ports als Eingänge geschaltet.) Das Bitmuster für PAD müßte dann so aussehen: 0000 0001. Nur PA 0 soll ja Ausgang sein; um das zu erreichen, beginnen wir unser Programm so:

```
0200 A9 01    LDA # 01
0202 8D 01 17 STA 1701
```

Der Akku wird mit hex 01 geladen, was binär dem gewünschten Bitmuster 0000 0001 entspricht, und an PAD abgespeichert. Das restliche Programm, um die LED mit dem Schalter ein- oder ausschalten zu können, sähe z. B. so aus:

```
0205 AD 00 17 LDA 1700
0208 10 04    BPL 020E
020A A9 01    LDA # 01
020C D0 02    BNE 0210
020E A9 00    LDA # 00
0210 8D 00 17 STA 1700
0213 4C 05 02 JMP 0205
```

Das Programm weist einige Besonderheiten auf. An der Adresse 0205 wird PA in den Akku geladen. Das höchstwertige Bit von PA wird dabei (wie bei allen Ladebefehlen des 6502) in das Negativ-Flag des Status-Registers übernommen. Abhängig von ihm (BPL = Branch on Plus, Sprung bei N-Flag = 0) wird entweder 01 oder 00 in den Akku geladen und an PA gespeichert, so daß PA 0 auf 1 oder 0 gesetzt wird. Der Befehl BNE bei 020C ist hier ausnahmsweise ein unbedingter Sprung: Vor ihm wurde der Akku mit 01 geladen, so daß das Zero-Flag des Statusregisters rückgesetzt ist und die Sprungbedingung (Branch on Not Equal) stets erfüllt ist. (Bei anderen Mikrocomputern müssen nur die Adressen von PA und PAD geändert werden; beim AIM-65 z. B. auf A001 und A003.)





## Der BIT-Befehl

An der Adresse 0205 haben wir die Daten von PA (1700) in den Akku geladen, obwohl wir sie gar nicht weiterverarbeiten wollen. Vielmehr wollten wir ja nur das N-Flag im Statusregister abhängig von der Schalterstellung an PB 7 setzen oder rücksetzen; und dafür läßt sich auch der Befehl BIT einsetzen. Er hat auf das N-Flag die gleiche Wirkung wie der LDA-Befehl, aber ohne den Akkuinhalt zu ändern, was manchmal recht nützlich ist.

Außerdem beeinflusst er das Zero-Flag abhängig von einer AND-Operation zwischen Akku und der angesprochenen Adresse. Legen wir einmal den Schalter nicht an PA 7, sondern z. B. PA 3, so wird gleich deutlich, wie man das verwenden kann:

0200 A9 01	LDA # 01
0202 8D 01 17	STA 1701
0205 A9 08	LDA # 08
0207 2C 00 17	BIT 1700
020A D0 04	BNE 0210
020C A9 01	LDA # 01
020E D0 02	BNE 0212
0210 A9 00	LDA # 00
0212 8D 00 17	STA 1700
0215 4C 05 02	JMP 0205

Der BNE-Befehl bei 020A verzweigt hier in Abhängigkeit der AND-Operation zwischen Akku und PA, aber ohne den Akku zu verändern. Der Akku enthält während des BIT-Befehls hex 08, also das Bitmuster 0000 1000. An der Stelle von PA 3 ist also eine 1. Je nach dem Ergebnis (08 oder 00) der AND-Operation kann das Z-Flag also gesetzt oder rückgesetzt werden.

Das gleiche Programmierproblem könnte man auch mit dem AND-Befehl lösen; der Operationscode 2C müßte dann durch 2D ersetzt werden. Der einzige Unterschied, der hier allerdings keine Rolle spielt, ist, daß nach dem AND-Befehl im Akku tatsächlich das Ergebnis der Operation steht, während der BIT-Befehl nur die Status-Flags beeinflusst.

Viele Mikrocomputer kennen keinen BIT-Befehl. Bei ihnen wird das „Maskieren“ des gerade abzufragenden Ports deshalb mit einem AND-Befehl vorgenommen, so daß im Akku nur die gerade interessierenden Bits des Ports übrigbleiben.

Bei den Prozessoren 8080 und Z-80 sind nicht Memory-Map-I/O-Bausteine üblich, sondern die Ein- und Ausgabe geschieht mit speziellen I/O-Befehlen der CPU, die dann an die Ein-Ausgabe-Bausteine ein Chip-Select-Signal schickt. Das hat allerdings den Nachteil, daß die I/O-Ports sich nicht in beliebige arithmetische Operationen (AND, OR, ADC usw.) einbeziehen lassen, da die I/O-Befehle nur einen Datentransfer erlauben. Andererseits hat

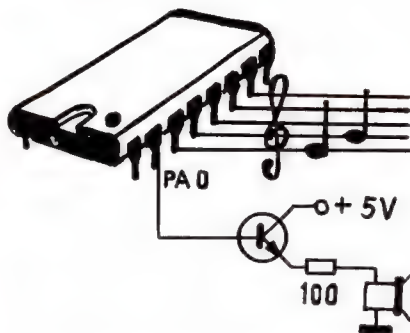
diese Methode den Vorteil, für I/O-Operationen nur 2-Byte-Befehle zu benötigen, wenn man sich auf max. 255 Ports beschränkt; bei Memory-Map-I/O-Anordnungen sind dagegen normalerweise 3-Byte-Befehle nötig, da eine 2-Byte-Adresse angegeben werden muß. Die Erfahrung zeigt jedoch, daß die zu erwartende Geschwindigkeitseinbuße des Memory-Map-Verfahrens in der Praxis nicht existiert.

## Hat man Töne?!

Eine recht nette Anwendung eines Mikrocomputers ist es, an einem I/O-Port einen Ton bestimmter Frequenz zu erzeugen. Sehen wir uns einmal ein einfaches Programm an, das dieses tut.

0200 A9 01	LDA # 01
0202 8D 01 17	STA 1701
0205 A2 60	LDX # 60
0207 CA	DEX
0208 D0 FD	BNE 0207
020A EE 00 17	INC 1700
020D 4C 05 02	JMP 0205

Das Programm, das für die I/O-Belegung des KIM-1 ausgelegt ist, erzeugt an PA 0 einen Ton, dessen Höhe abhängig von dem Inhalt der Speicherzelle 0206 ist. Damit wird nämlich das X-Indexregister geladen und in einer Verzögerungsschleife heruntergezählt, bis es Null ist.



Der INC-Befehl dient dazu, den Zustand von PA 0 umzudrehen. Bekanntlich wird beim Inkrementieren eines Byte jedesmal das niederwertigste Bit komplementiert; die höherwertigen interessieren uns hier nicht. (Hier zeigt sich der Vorteil des Memory-Map-Verfahrens: Die Methode mit dem INC-Befehl ist bei Prozessoren, die besondere Ein- und Ausgabe-Befehle haben, nicht möglich.)

Mit Hilfe des Programmier-Handbuches läßt sich errechnen, wie hoch die erzeugte Frequenz je nach dem Inhalt des X-Registers ist. Das KIM-1-Handbuch gibt ein Beispiel, wie man die erzeugte Frequenz mit Hilfe von Schaltern verändern kann – dabei wird ein I/O-Port abgefragt.

Dieses Tonerzeugungs-Programm ist ein typisches Beispiel dafür, daß man nicht alle Software-Probleme in einer höheren Programmiersprache wie BASIC lösen kann. Während für einen BASIC-Befehl u.U. mehrere Millisekunden gebraucht werden, begnügen sich die Maschinenbefehle mit wenigen Mikrosekunden. Die genauen Befehls-Ausführungszeiten hängen vom Prozessor, von der Adressierungsart und von der CPU-Taktfrequenz ab und können in den Software-Handbüchern und Befehlstabellen der CPU-Hersteller nachgeschlagen werden. Die Zeiten werden meist als Zyklus-Zahlen angegeben, was beim 6502 und 1 MHz Taktfrequenz (AIM-65, KIM-1, SYM-1, PC-100) gleichzeitig die Befehlszeit in Mikrosekunden ergibt.

Läßt man die Verzögerungsschleife (0205...0208) weg, so kann man mit Hilfe der nun maximal erzeugbaren Frequenz an einem I/O-Port einfache Vergleiche der Geschwindigkeit unterschiedlicher Prozessoren anstellen. Bei 1 MHz Taktfrequenz ergeben sich mit dem 6502 etwa 55 kHz. (Derzeit existiert die CPU 6502 in unterschiedlichen Versionen für max. 2 MHz Taktfrequenz.)

## Ist das alles?

Wir haben inzwischen eine Reihe von Adressierungsarten und Befehlen kennengelernt. Natürlich soll das Papier, das Sie gerade vor sich haben, nicht das Programmier-Handbuch Ihres Mikroprozessors ersetzen. Viele Befehle, die dort aufgeführt sind, bedürfen auch gar keiner weiteren Erklärung, weil ihre Bedeutung schon aus ihrer Bezeichnung hervorgeht. Sinn der Sache ist es hier vielmehr, jene Dinge etwas näher zu beleuchten, die in manchen Programmierbüchern als selbstverständlich vorausgesetzt werden und dann zu Verständnisschwierigkeiten, ja zur Entmutigung führen.

Dabei ist das Programmieren in Maschinensprache recht einfach, wenn man einmal die notwendigsten Grundbegriffe und die wichtigsten Befehle „seines“ Mikroprozessors kennt. Voraussetzung ist lediglich die Bereitschaft zum streng logischen, konsequenten Denken in kleinsten Schritten – nämlich in CPU-Befehlen.

Wenn Sie bis hierher mitgemacht und die Programme auch verstanden haben, so dürfte es Ihnen keine Schwierigkeiten bereiten, einfache Programme selbst zu schreiben, zum Beispiel eine 16-bit-Addition oder ein Programm, das die Anzeige auf dem KIM-1 blinken läßt.

Die ersten Programme, die man schreibt, sind immer ziemlich unständlich. Später stellt man fest, daß alles viel einfacher gegangen wäre, wenn man geeignetere Befehle angewandt



hätte, mit denen man erst nach einiger Zeit vertraut wurde. Nicht selten kann man seine „Erstlingswerke“ um ein Drittel verkürzen.

Etwas, was man tatsächlich erst nach einigem Überlegen und einiger Erfahrung versteht, ist die indirekt indizierte Adressierungsart – eine nützliche Eigenart des 6502.

### Indirekt indizierte Adressierung

Wir haben bereits ein Programm gesehen, das einen bestimmten Speicherbereich mit Nullen auffüllt; es war der Bereich 0301...937F. Die damals gewählte Methode der indizierten Adressierung mit dem nur 8 bit langen X-Index-Register beschränkt den überstreichbaren Bereich auf maximal hex FF (dezimal 255)Byte, also genau eine „Page“. Was ist aber zu tun, um z. B. den Bereich 0200...03FF zu löschen, also das obere halbe KByte des KIM-1? Natürlich könnte man zwei solche Programme aneinanderketten, die jeweils eine Page löschen – bei größeren Speicherbereichen wird das schnell uneffektiv. (Andere Mikroprozessoren, z. B. der 6800, besitzen ein 16-bit-Indexregister, das dieses Problem löst – allerdings eben nur eines...)

Beim 6502 bedient man sich zum Überstreichen größerer Speicherbereiche der „indirekt indizierten Adressierung“. Dabei werden zwei aufeinanderfolgende Zero-Page-Zellen als Indexregister verwendet, was praktisch einem 16-bit-Register entspricht. Dieser Trick ermöglicht es dem 6502, über 128 Register mit 16 bit Länge zu verfügen, denn die Zero Page umfaßt ja 256 Byte. Aus diesem Grunde ist es ungerecht, die wenigen internen Register der CPU 6502 mit dem umfangreicheren Registersatz von Prozessoren wie dem Z-80 zu vergleichen. Die scheinbare Schwäche wird durch eine Vielzahl nützlicher Adressierungsarten wettgemacht. Das Programm zum Löschen des Bereiches 0200...03FF sieht so aus:

```
0000 A9 00    LDA # 00
0002 85 E0    STA E0
0004 A9 02    LDA # 02
0006 85 E1    STA E1
0008 A0 00    LDY # 00
000A A9 00    LDA # 00
000C 91 E0    STA (E0),Y
000E E6 E0    INC E0
0010 D0 02    BNE 0014
0012 E6 E1    INC E1
0014 A5 E1    LDA E1
0016 C9 04    CMP # 04
0018 D0 F0    BNE 001A
0019 4C ...   JMP Monitor
```

Was geschieht hier? Zunächst werden die Zero-Page-Zellen 00E0 und 00E1 mit der Anfangsadresse des zu löschenden Bereichs geladen, wobei das höherwertige Adressenbyte in 00E1 steht. Akku und Y-Register werden

dann mit 00 geladen – und jetzt wird der Akkuinhalt an diejenige Adresse gespeichert, die sich aus dem Inhalt der Adressen 00E0/00E1 plus dem Y-Index ergibt. Zunächst ist dies 0200. Die Befehlsfolge an den Adressen 000E...0012 erhöht die 16-bit-Adresse im Zellenpaar 00E0/00E1. Schließlich wird noch durch Abfragen des höherwertigen Adressenteils in 00E1 abgefragt, ob bereits Page 4 erreicht ist; wenn nein, wiederholt sich das Spiel für die nächste Adresse (0201), wenn ja, erfolgt ein Rücksprung zum Monitorprogramm, z. B. an die Adresse 1C4F beim KIM-1.

Dieses Programm ist eines der wenigen Beispiele, die die Programmierung des 6502 ein wenig umständlich erscheinen und die Herzen der Besitzer anderer Prozessoren höher schlagen lassen.

Von der Möglichkeit, zusätzlich einen Adressenversatz durch das Y-Register zu erreichen, wurde hier kein Gebrauch gemacht – Y bleibt während des ganzen Programmes Null. Übrigens können geübte Programmierer dieses Programm ohne weiteres um zwei oder mehr Byte verkürzen – versuchen Sie's mal! (Ein Hinweis: Man kann zu Beginn auch das Y-Register verwenden, um die Null in 00E0 zu laden.)

Die indirekt indizierte Adressierung gibt es beim 6502 nur in Verbindung mit dem Y-Register. Dem X-Register ist eine Adressierungsart vorbehalten, die nicht minder kompliziert ist.

### Indiziert-indirekte Adressierung

Die indiziert-indirekte Adressierung wird nicht so häufig gebraucht und soll daher nur kurz vorgestellt werden. Ein Beispiel hilft, diese komplizierte Adressierungsart zu verstehen. Der Befehl STA (E0,X), der dem vorhin gebrachten Beispiel etwas ähnelt, tut folgendes:

Ist der Inhalt des X-Registers z. B. 07, so wird zunächst einmal die Summe von E0 und 07 errechnet, sie ergibt E7. Dann wird der Akku an jene Adresse gespeichert, die im Zellenpaar 00E7/00E8 steht.

Dieser Befehl (hex 91 E0) kann alternativ in unserem letzten Programm verwendet werden, wenn man das X-Register statt dem Y-Register vorher auf Null setzt. In diesem Fall (und nur dann) ist die Wirkung von indiziert-indirekter und indirekt indizierter Adressierung identisch.

Während man anfangs mit vielen Adressierungsarten noch nichts so recht anzufangen weiß, ärgert man sich nach einiger Zeit darüber, daß nicht für jeden Befehl jede Adressierungsart zur Verfügung steht. Leider haben die Entwickler des 6502 hier ein wenig gespart. Nur der 16-bit-Mikroprozessor 68 000 von Motorola kann hier als Mu-

sterknabe betrachtet werden: Für alle Befehle ist jede (sinnvolle) Adressierungsart vorhanden.

### Interrupt-Verarbeitung

Bisher waren unsere Programme auf Neuhochdeutsch „straightforward“ – sie bearbeiteten irgendwelche Daten, führten ab und zu Sprungbefehle oder Unterprogramme aus, ließen sich sonst aber durch nichts aus der Ruhe bringen.

Halt – nur durch eines natürlich: Wenn wir auf die Reset-Taste unseres Mikrocomputers drückten, hörte unser Anwenderprogramm mit der Ausführung seiner Befehle auf, und es war wieder möglich, mit der Tastatur und dem Display Adressen und Daten zu ändern. Die Reset-Taste bewirkt also einen gewaltsamen, durch Hardware hervorgerufenen Sprung zum Monitorprogramm, der durch nichts, aber auch gar nichts verhindert werden kann. Der Sprung wird dadurch ausgelöst, daß ein bestimmter Pin der CPU – die Reset-Leitung – durch Tastendruck kurz auf log. 0 (Low) gelegt wird.

Was passiert hier? Sofort, wenn die Reset-Leitung auf Low geht, lädt die CPU ihren internen Programmzähler mit einer Adresse, die – hardwaremäßig festgelegt – beim 6502 in den Adressen FFFC und FFFD steht. (Verwechseln Sie hier nicht Adresse mit Adresse – die Zellen FFFC/FFFD enthalten zwei Byte, die die CPU als neue Adresse betrachtet.) Dann fährt die CPU mit der Ausführung der Befehle am neuen Programmzählerstand fort, nämlich mit den Befehlen des Monitorprogramms. Das ist alles, was ein „Reset“ in der CPU bewirkt. Eine solche hardwaremäßige, gewaltsame Programm-Unterbrechung bezeichnet man allgemein als Interrupt. Der 6502 kennt drei Interruptarten: Reset (siehe oben), NMI und IRQ.

#### NMI

NMI heißt „non-maskable interrupt“ – das hat er eigentlich mit dem Reset gemeinsam: Er läßt sich softwaremäßig nicht verhindern. In einem unterscheidet er sich aber vom Reset: Wie bei einem Unterprogramm wird die Rücksprungadresse und hier zusätzlich noch der Inhalt des Statusregisters auf den Stack „gerettet“. Der Sprung beim Auftreten des NMI erfolgt hier über den Inhalt des Zellenpaares FFFA und FFFB. Stehen dort z. B. die Daten 05 A3, so springt der Prozessor an die Adresse A305. Dort steht dann irgendeine Routine, die irgend etwas bearbeitet. Der Trick: Weil die Rücksprungadresse auf dem Stack abgespeichert wurde, kann man mit dem Befehl RTI wieder in das unterbrochene Programm zurückkehren, als wäre nichts geschehen – sogar das Statusregister hat dann wieder seinen alten Inhalt, weil RTI im Gegensatz zu RTS auch den Prozessorstatus vom Stack zurückholt.



## IRQ

Eine dritte Interrupt-Leitung der CPU heißt „IRQ“. Dieser Interrupt unterscheidet sich vom NMI dadurch, daß er per Software verhindert werden kann. Dafür gibt es den Befehl SEI = Set Interrupt Disable Flag. Vom Interrupt-Flag im Status-Register hängt es ab, ob ein Low-Signal am IRQ-Eingang der CPU tatsächlich zu einem Interrupt führt oder nicht. Ebenso wie beim NMI werden beim Auftreten des „Interrupt Request“ Rücksprungadresse und Statusregister auf den Stack gerettet. Der IRQ-„Vektor“ steht an den Adressen FFFE und FFFF.

### Der Break-Befehl

Der IRQ-Interrupt kann softwaremäßig simuliert werden, nämlich mit dem BRK-Befehl (Operationscode 00). Auch hier werden Statusregister und Programmzähler auf den Stack gerettet. Der Break-Befehl ist beim Durchtesten von Programmen recht nützlich, da es mit ihm möglich ist, das Programm an beliebigen Stellen anzuhalten, indem man BRK einfügt.

### Stackpointer-Korrektur

Erfolgt nach einem Interrupt oder nach dem Break-Befehl kein Rücksprung mit RTI, so muß man den Stackpointer wieder korrigieren (s.a. „Unterprogramme“), da er ja nicht mehr in seiner Normallage steht (FF). Dies kann wieder mit der Befehlsfolge

LDX # FF

TXS

erfolgen. Vergißt man dies, so können wiederum wichtige Daten oder Programmteile im Stackbereich (beim 6502 in Page 1) überschrieben werden. Bei Prozessoren, deren Stackpointer nicht 8, sondern 16 bit lang ist, kann das zu chaotischem Verhalten führen.

### Interrupt-Quellen

Was kann überhaupt einen Interrupt auslösen? Nun, prinzipiell alles, was in der Lage ist, an einen Interrupt-Pin der CPU einen Low-Impuls zu liefern. Beim KIM-1 kann das z. B. die STOP-Taste sein, oder aber, wenn man den Timer-Ausgang PB7 mit NMI oder IRQ verbindet, auch ein auf dem Mikrocomputer befindlicher Timer, der mit einer bestimmten Zeit geladen werden kann und nach Ablauf dieser einen Interrupt auslöst. Eine hübsche Anwendung dieses Timer-Interrupts ist u.a. eine Software-Uhr, die für den KIM-1 in FUNK-SCHAU 1979, Heft 11, Seite 657 beschrieben ist.

### Die Register müssen auf den Stack

Wie bei manchen Unterprogrammen kann es notwendig sein, die CPU-Register beim Auftreten von Interrupts auf den Stack zu „retten“. Da im Gegensatz

zu Unterprogrammen aber nicht vorhergesagt werden kann, wo im Hauptprogramm der Interrupt auftritt – das kann bei jedem beliebigen Befehl geschehen – muß man in der Interrupt-Routine alle Register „retten“, die in ihr verändert werden. Normalerweise ist dies der Akku, in manchen Fällen auch die Indexregister X und Y. Ein Beispiel: Beim Auftreten eines Interrupts soll der Inhalt des Ports A beim KIM-1 komplementiert werden (Port A muß dabei natürlich als Ausgang geschaltet sein; nur dann können alle seine Bits komplementiert werden).

Die Interrupt-Routine muß dann z. B. so aussehen:

0280	48	PHA
0281	AD 00 17	LDA 1700
0284	49 FF	EOR # FF
0286	8D 00 17	STA 1700
0289	68	PLA
028A	40	RTI

Das Komplementieren wird hier mit dem EOR-Befehl (Exklusiv-Oder) im Akku erreicht. Damit der Akku-Inhalt bei der Rückkehr in das Hauptprogramm (hier nicht aufgelistet) nicht verändert ist, wird der Akku zu Beginn der Interrupt-Routine auf den Stack gerettet und vor ihrem Ende zurückgeholt. Die obige Routine kann beim KIM-1 leicht durch Drücken der STOP-Taste prüfen, wenn der NMI-Vektor in den Zellen 17FA und 17FB auf die Adresse 0280 zeigt, d.h. in 17FA muß 80 und in 17FB muß 02 stehen.

### Indirekte Sprünge

In den meisten Mikrocomputern wird der NMI- oder IRQ-Vektor nicht wirklich aus FFFA/FFFB bzw. FFFE/FFFF geholt, sondern aus Zellen im RAM-Bereich, zu denen ein indirekter Sprung

aus dem Monitorprogramm führt. Die Vektoren in FFF... zeigen dabei auf diesen indirekten Sprungbefehl im Monitor-ROM. Beim KIM-1 sieht das so aus: Der Vektor in FFFA/FFFB zeigt auf die Adresse 1C1C. Dort wiederum, also im Monitor-ROM, steht der Befehl JMP (17FA), hex 6C FA 17. Das Programm springt daraufhin an die Adresse, die der Anwender in die RAM-Zellen 17FA und 17FB geschrieben hat.

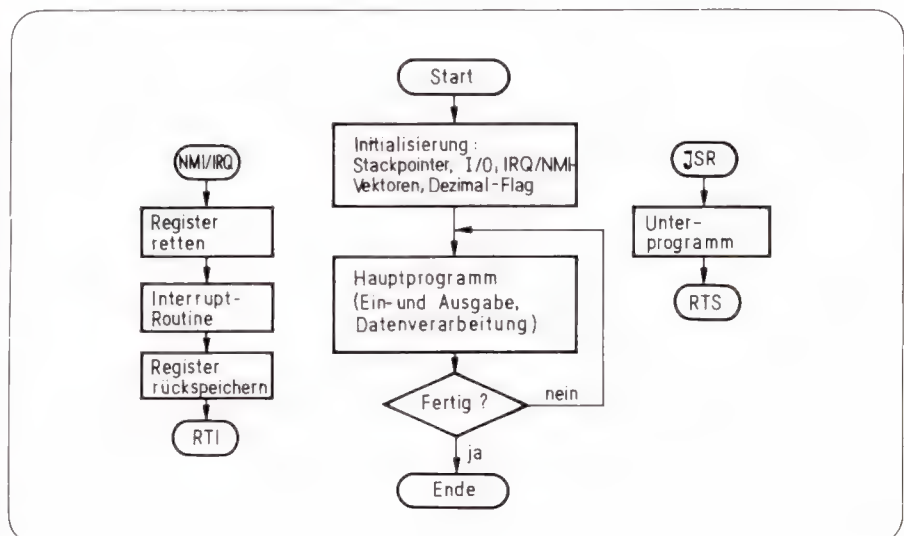
## Über das Schreiben von Programmen

Mittlerweile haben wir schon das notwendige Rüstzeug, um fremde Programme ohne große Schwierigkeiten verstehen zu können; das ist oft nur eine Frage der Konzentration und Geduld.

Längere Programme selbst zu schreiben, erfordert schon etwas Erfahrung. Jedem Anfänger ist zu raten, erst einmal fertige Programme zu betrachten, eventuell gezielt Änderungen vorzunehmen und deren Wirkungen zu beobachten. Dadurch lernt man auch am schnellsten die unterschiedlichen Befehle und Adressierungsarten kennen. Einen guten Anhaltspunkt bieten oft die gut kommentierten Monitor-Programm-Listings in den Systemhandbüchern, wie etwa der Mikrocomputer KIM-1, SYM-1 oder AIM-65.

Das Schreiben von Programmen besteht grundsätzlich aus vier wichtigen Abschnitten:

1. Exakte Definition der Problemstellung
2. Umsetzen des Problems in eine logische Folge (z. B. Flußdiagramm)
3. Erstellen des Maschinenprogrammes
4. Testen und Korrigieren des Programms





Entgegen der Annahme vieler Neulinge beansprucht nicht etwa Punkt 2 oder 3 am meisten Zeit, sondern Punkt 4. Kein Programm läuft auf Anhieb!

Falsche Zeitvorstellungen existieren auch oft über Punkt 1. Eine exakte Problemdefinition vor dem Schreiben des Programms erspart umständliche, später kaum noch durchführbare Änderungen im Programmablauf. Dazu gehört speziell die Berücksichtigung von Randbedingungen, die z. B. bei ungewöhnlichen und unzulässigen Eingaben durch den späteren Programm-Benutzer oder durch u.U. unwahrscheinliche Konstellationen von Daten auftreten. Was passiert, wenn der Computer eine Zahl erwartet, der Benutzer aber den Buchstaben Z drückt? Wie soll der Computer reagieren, wenn zu viele Ziffern eingegeben werden? Was geschieht, wenn ein Kontakt an einem Eingangs-Port prellt? Alle diese Dinge gehören zu einer exakten Definition der Problemstellung.

Mit der Problemdefinition haben wir natürlich das Problem noch nicht gelöst. Oft muß man auch in das Erstellen des Flußdiagramms noch eine Menge Hirnschmalz investieren, da das Problem ja meist nicht als logische, zeitlich gestaffelte Instruktionsfolge vorliegt, sondern eher in einer chaotischen, unüberschaubaren Form.

Dann ist es soweit: Aus dem Flußdiagramm muß irgendwie der hexadezimale Maschinencode gewonnen werden. Bei einfachen Systemen wie dem KIM-1 oder SYM-1 bedeutet das, daß man – wenn man die Befehle noch nicht auswendig kennt – jeden Operationscode in einer Tabelle nachsehen muß. Der AIM-65 hat eine solche Tabelle eingebaut, nämlich im ROM, und übernimmt die Übersetzung der mnemonischen Befehle in den Hex-Code selbst; z. B. speichert er A9 ins RAM, wenn man die Tasten LDA # drückt. Einen solchen Übersetzer nennt man Assembler – die Rückübersetzung heißt dementsprechend Disassemblierung.

Für das Schreiben von Maschinenprogrammen gibt es einige Regeln, die man sich merken sollte:

1. Verwenden Sie bei Sprungbefehlen möglichst immer die relative Adressierung. Das spätere Verschieben von Programmteilen wird damit problemlos, auch wenn man keinen Assembler mit symbolischer „Label“-Adressierung hat.
2. Vermeiden Sie unbedingt Programme, die sich selbst ändern, d.h. die selbst Operationscodes innerhalb des Programmes ändern. Erstens läuft das Programm dann nur im RAM und nie im EPROM, zweitens weiß man später, wenn man das Programm ausdruckt, nie genau, in welchem „Zustand“ es gerade war.
3. Wählen Sie keine „krumme“ Startadresse, die sich niemand merken kann, sondern eine gerade, wie z. B. 0000 oder 0200.
4. Meiden Sie den Stackbereich (0100...01FF)! Wenn ein Programm wegen eines kleinen Fehlers einmal „Harakiri“ läuft, ist der Stackbereich der erste, der ihm zum Opfer fällt. Dann ist man erst einmal damit beschäftigt, das Programm neu einzutippen.
5. Machen Sie sich eine genaue Aufstellung, welche Zellen außerhalb des Programms noch belegt werden, z. B. Zero-Page-Adressen. Und überhaupt: Versuchen Sie schon während des Programmierens, Ihre Software so zu dokumentieren, daß Sie das Programm auch noch nach einem Jahr verstehen können!
6. Wenn das Programm Interrupt-Routinen enthält, so sollte man es nicht dem Erinnerungsvermögen des Benutzers überlassen, vor dem Start

die IRQ- oder NMI-Vektoren zu setzen. Das sollte besser gleich nach dem Start des Hauptprogrammes automatisch geschehen.

7. In der endgültigen Form sollten die einzelnen Programmteile nicht gleichmäßig über den gesamten RAM-Bereich des Mikrocomputers verteilt sein, sondern aneinandergehängt werden, damit man das Komplettdprogramm auf einmal z. B. auf eine Kassette aufzeichnen oder auch in ein EPROM laden kann.
8. Unterprogramme des Monitor-ROM sollten nur dort verwendet werden, wo es wirklich notwendig ist. Andernfalls ist eine Adaption des Programmes an andere Mikrocomputersysteme oft unmöglich.
9. Vermeiden Sie möglichst Operationscodes, die zwar offensichtlich funktionieren, aber nicht im Programmierhandbuch des CPU-Herstellers verzeichnet sind. Diese Codes funktionieren u.U. bei CPUs anderer Hersteller nicht oder führen bei bestimmten Datenkonstellationen zu Problemen.
10. Legen Sie sich ein Telefon mit abschaltbarer Klingel zu. Zum Programmieren braucht man Konzentration!

Mit diesen „zehn Geboten für den Programmierer“ wollen wir unseren Streifzug durch die Maschinenprogrammierung beenden. Sicher sind Sie jetzt in der Lage, Programme, die in Zeitschriften und Büchern veröffentlicht sind, zu verstehen – und manchmal sogar zu verbessern.

#### Stichworte zum Inhalt

Maschinenprogrammierung, Einführung, Grundlagen, hexadezimal, ASCII, Register, Adressierungsarten, Monitorprogramm, Stack, Memory Map, Interrupt.

## Die „unterstrichenen“ BASIC-Listings

An mehreren Stellen in diesem Heft finden Sie Programme in BASIC, die mit einem Centronics-779-Drucker aufgelistet wurden. Dieser Drucker ist zwar in der Lage, alle PET-Grafik-Zeichen wiederzugeben, bei der inversen Zeichendarstellung macht

er aber nicht mehr mit und druckt dann einfach in der normalen Zeichenform. Wenn Sie in den Listings einzelne Zeichen unterstrichen finden, so bedeutet das, daß diese Zeichen auf dem PET-Bildschirm invers erscheinen; meist handelt es sich

dabei um Steuerzeichen, z. B. Cursorbewegungen innerhalb eines PRINT-Befehls. Die Inversdarstellung wurde von den Drucker-Entwicklern nicht realisiert, weil sie den Druckknopf zu sehr abnützen würde; Grafikzeichen sind möglich.



Wenn man drei Spezialisten fragt, welcher Mikroprozessor heute der „beste“ sei, erhält man mindestens vier unterschiedliche Auskünfte. Kein Wunder: Mikroprozessoren wurden von Leuten entwickelt, die ganz bestimmte Aufgabenstellungen im Auge hatten, und müssen daher nicht unbedingt für alle anderen Aufgaben auch optimal sein. Der folgende Beitrag geht auf die wichtigsten Eigenschaften heutiger CPUs ein.

# Der beste Mikroprozessor

## Überlegungen zur CPU-Auswahl

### 8- und 16-bit-Prozessoren

In der letzten Zeit kamen einige 16-bit-Prozessoren heraus. Gegenüber dem bisherigen 8-bit-Standard erlauben sie wegen ihrer breiteren Wortlänge eine schnellere Verarbeitung mehrstelliger Zahlen, was bei arithmetischen Aufgabenstellungen interessant sein kann. Bei der Textverarbeitung sind die Vorteile der 16-bit-CPU's nicht so groß, weil ein ASCII-Zeichen nun einmal nur 7 bit benötigt.

Der Vorteil einer höheren Arbeitsgeschwindigkeit ergibt sich aber nur bei solchen Prozessoren, die auch alle 16 bit gleichzeitig auf den Datenbus geben können, also 16 Datenbus-Pins aufweisen. Das Multiplexen von Adressen- und Datenbus (TMS 9980, 8086) macht den Vorteil einer höheren Geschwindigkeit schnell zunichte. Allerdings besitzen 16-bit-CPU's gegenüber ihren 8-bit-Vorfahren meist einen wesentlich komfortableren Befehlssatz.

Bei den heutigen Hobbycomputer-Anwendungen ist der höhere Hardware-Aufwand im Vergleich zu 8-bit-CPU's aber kaum vertretbar. Daß z. B. Texas Instruments einen Personal Computer mit 16-bit-CPU herausbrachte, hängt weniger damit zusammen, daß die Wortbreite von 16 bit notwendig ist, sondern eher damit, daß TI keine 8-bit-Prozessoren fertigt.

Ohne Zweifel werden also die 8-bit-Prozessoren nicht von 16-bit-Typen „abgelöst“ und haben nach wie vor ihre Berechtigung. Die Vorteile der 16-bit-CPU's sind vor allem bei zeitkritischen Industrieanwendungen interessant.

### Befehlssatz

Die Zahl der zur Verfügung stehenden CPU-Befehle oder Operationscodes zur Beurteilung der Leistungsfähigkeit eines Mikroprozessors heranzuziehen, ergäbe ein völlig falsches Bild. Im Gegenteil: Der ideale Mikroprozessor besitzt relativ wenige, dafür aber – z. B. wegen komfortabler Adressierungsarten – leistungsfähige Befehle. (Es ist interessant, daß der 16-bit-Prozessor 68000 gegenüber seinem 8-bit-Vorläufer 6800 nicht etwa mehr, sondern 11 Befehle weniger besitzt!)

Ein Mikroprozessor kann nur dann optimal programmiert werden, wenn der Benutzer alle seine Befehle auswendig kennt; andernfalls geht der Vorteil des umfangreichen Befehlssatzes verloren. So ist es kein Wunder, daß man länger braucht, um auf dem Z-80 fit zu werden, als auf dem wesentlich komfortableren 68000. Entscheidend ist dabei die Zahl der unterschiedlichen „Mnemonics“, also der Befehls-Kurzbezeichnungen, die man lernen muß. (Die Zahl der Operationscodes ist höher als die der Mnemonics, da ja viele Befehle mehrere Adressierungsarten er-

Typen hat Befehle für Multiplikation und Division, und die beim Z-80 so gerühmten Blocksuchbefehle lassen sich auf den anderen Prozessoren meist mit wenigen Befehlen implementieren.

### Geschwindigkeit

Betrachtet man den Zeitbedarf für bestimmte Befehle der heute gängigen Prozessoren, so sind die Unterschiede nicht allzu groß, wie ebenfalls Tabelle 1 am Beispiel einer binären und einer dezimalen 8-bit-Addition des Akku mit dem Inhalt einer bestimmten Speicherzelle und mit Übertrag (Carry) zeigt. Bei

Tabelle 1: Vergleich einiger üblicher Prozessoren

	8080A	Z-80	6800	6502
Mnemonics	78	67	72	56
Zyklen f. bin. Addition	7	7	3	3
Additionszeit/µs	3.85	3.08	3	3
Zeit f. dezim. Add.	5.5	4.4	5	3
Adressierungsarten beim Addieren	3	6	4	8

lauben, die aus der Schreibweise des Arguments, das dem Befehl folgt, hervorgehen.) Tabelle 1 gibt die Anzahl der Mnemonics bei unterschiedlichen Prozessortypen an. Allerdings wäre es natürlich auch falsch, zu sagen, daß der Mikroprozessor mit den wenigsten Mnemonics in jedem Falle der „beste“ ist. Immerhin geht aus der Tabelle hervor, daß keine Proportionalität zwischen Prozessorqualität und Befehlssatz besteht. Übrigens hat der Z-80 nur scheinbar weniger Befehle als der 8080; in Wirklichkeit ist es umgekehrt, weil Zilog eine andere Assembler-Syntax verwendet, die die Unterscheidung zwischen einigen ähnlichen Befehlen in die Adressierungsart und nicht in die mnemonische Befehlsform hineinpackt.

Der Befehlssatz der einzelnen, in der Tabelle 1 aufgeführten Prozessortypen ist nicht so groß, wie man vielleicht glauben könnte, wenn man die Datenblätter der Hersteller liest. Keiner dieser

der dezimalen Addition schneidet der 6502 am besten ab, weil er sich mit SED fest in einen dezimalen Modus umschalten läßt (SED wurde hier nicht mitgerechnet, weil dieser Befehl ja nur einmal am Anfang eines Programms zu stehen braucht).

Aber auch bei der binären Addition liegen die Typen 6800 und 6502 wegen der Zero-Page-Adressierungsart vorn, bei der nur eine 8-bit-Adresse (Bereich 0000...00FF) angegeben werden muß. Die dezimale Addition erfordert bei den Prozessoren 8080A, Z-80 und 6800 zwei Befehle (ADC und DAA).

Doch macht eine Mücke noch keinen Sommer, und in bestimmten Fällen können die Prozessoren ganz anders im Rennen liegen, etwa beim Absuchen eines Speicherbereichs nach einem bestimmten Byte – hier läge der Z-80 vorn. Erst längere Programme geben Aufschluß über die reale Geschwindigkeit einer CPU; BASIC-Interpreter sind ein guter Anhaltspunkt. Und hier ist die



Reihenfolge meist: 6502, Z-80, 6800, 8080. Selbstverständlich geht aber auch hier die Rechengenauigkeit und die Ausgefeiltheit der Programmierung mit ein – letztere ist allerdings nicht zuletzt ein Resultat einfacher oder komplizierter Programmierbarkeit einer CPU. Auch sollte man nicht übersehen, daß in vielen Anwendungen die Geschwindigkeit eine nur untergeordnete Rolle spielt.

Der Vorteil schneller Prozessoren ist oft darin zu sehen, daß sie bestimmte Hardware-Teile des Systems durch Software ersetzen können, z. B. bei der Serien-Parallel-Wandlung, der Erzeugung von Tonfrequenzen oder auch der Analog-Digital-Wandlung.

## Software

In der Industrie werden die Kosten bei der Entwicklung eines Mikrocomputer-System längst im wesentlichen von den notwendigen Programmen und nur zu einem geringen Teil von der Hardware bestimmt. Man wird sich daher meist einen Prozessor aussuchen, mit dem ein Mitarbeiter bereits Erfahrung besitzt und für den vielleicht schon fertige, häufige gebrauchte Programmteile (Makros, Unterprogramme, Moduln) existieren, z. B. aus Veröffentlichungen in Zeitschriften und Büchern (sehr preiswert) oder vom CPU-Hersteller (sehr teuer).

Für den Hobby-Programmierer gelten ähnliche Überlegungen. Gerade in der Lernphase wird man sich an fertigen Programmen orientieren, um übliche Programmierkniffe kennenzulernen. Auch wird kaum jemand auf die Idee kommen, sich mühsam z. B. einen Disassembler, einen Assembler oder einen BASIC-Interpreter selbst zu schreiben, wenn er solche Programme nur aus einer Zeitschrift abzutippen braucht. Jedes Programm von Grund auf selbst zu schreiben, wäre selbst für den Hobbyisten ein kaum vertretbarer Zeitaufwand.

Natürlich findet man in den einschlägigen Zeitschriften und Büchern nicht für jeden Prozessor gleich viele Programme; weitaus an der Spitze liegt derzeit der Typ 6502, in einigem Abstand folgen die CPUs 8080, Z-80 und 6800. Andere Prozessoren konnten sich auf dem Personal-Computer-Markt bisher nicht durchsetzen. Allein in der FUNKSCHAU fanden sich bisher für den 6502 mehr Programme als für alle anderen Typen zusammen.

Der Grund für die Verbreitung eines bestimmten Prozessors ist nur in zweiter Linie sein ausgefeilter Befehlssatz oder seine hohe Geschwindigkeit. Entscheidend ist vielmehr, ob preisgünstige und doch komfortable fertige Mikrocomputer mit einer bestimmten CPU existieren.

**Tabelle 2: Vor- und Nachteile einiger Prozessoren**

Typ	Vorteile	Nachteile
8080A	Viel Software erhältlich; viele Peripheriebausteine	Kein interner Taktgenerator; nur zwei Interrupt-Ebenen; mehrere Betriebsspannungen; externe Steuerlogik notwendig
Z-80	8080-software-kompatibel; Refresh f. dyn. RAMs; zwei 16-bit-Indexregister; Blocksuch- und Einzelbit-Befehle; zwei Registersätze; komfortable Interrupt-Verarbeitung	Schwer erlernbarer, unübersichtlicher Befehlssatz; mehrere 4-Byte-Befehle; hoher CPU-Preis
6800	Leicht erlernbarer Befehlssatz; zwei Akkus, 16-bit-Indexregister	Nur ein Indexregister; z. T. umständlicher Transfer der CPU-Register; kein Parity-Flag
6502	Leicht erlernbarer Befehlssatz; zwei Indexregister; 13 Adressierungsarten; sehr einfacher Steuerbus; Dezimal-Modus; geringer Preis; sehr viel Software erhältlich, universelle Peripheriebausteine	Kein Parity-Flag; DMA komplizierter als bei anderen CPUs; nur 8-bit-Indexregister; Stackpointer nur 8 bit lang
2650	Komfortables Statusregister; zwei 3-Register-Sätze + Akku	Nur 13 Adreßleitungen und 3-bit-Stackpointer; nur 1 Interrupt-Ebene; externer Takt nötig; wenig Software erhältlich
9980	16-bit-CPU mit Multiplikations- und Divisions-Befehlen. Register außerhalb der CPU im RAM	Nur 12 Adreßleitungen; Datenbus in 2x8 bit gemultiplext; nicht schneller als viele 8-bit-CPU's; wenig Software erhältlich
SCMP	Sehr billig; DLY-Befehl für Verzögerungen; Register für Serien-Parallelwandlung; 5 Statusbits per Hardware zugänglich	Nur 1- und 2-Byte-Befehle; umständliche Adressierung; keine Unterprogramme ohne Kniffe möglich; kein Parity-Flag; sehr langsam

## Auswahlkriterien

Alles bisher Gesagte klingt komplex und wenig „handgreiflich“. Tatsächlich ist der Vergleich der heute üblichen Prozessoren, wenn man objektiv und gerecht sein will, kaum auf ein paar Seiten möglich. Trotzdem sei hier eine kleine Entscheidungshilfe gegeben, nach welchen Kriterien der Hobby-Programmierer (und nur dieser!) eine für seine Zwecke sinnvolle Auswahl treffen kann.

### Operationscode-Tabelle

Ein guter Anhaltspunkt, ob ein Prozessor überschaubar in Maschinsprache zu programmieren ist, die Größe der Tabelle, die die hexadezimalen Operationscodes für die unterschiedlichen Mnemonics und Adressierungsarten angibt. Für den 6502 ist eine solche Tabelle in FUNKSCHAU 1979, Heft 11, zu finden; bei einer Schriftgröße von 2 mm ist sie knapp 6 cm x 10 cm „groß“. Der Hobbyist ist

gut beraten, wenn er Prozessoren vermeidet, bei denen sich eine solche Tabelle kaum auf einer DIN-A4-Seite unterbringen läßt.

### Hardware

Prozessoren, die dem Stand der Technik entsprechen, besitzen den Taktgenerator zum Erzeugen der zwei Steuertaktphasen auf dem CPU-Chip und brauchen nur eine Versorgungsspannung, nämlich +5 V. Alle CPU-Pins sollten TTL- oder/und CMOS-kompatible Signale aufweisen und mindestens eine TTL>Last treiben können.

In den Datenblättern der CPU-Hersteller sind meist nur die Vorteile bestimmter Prozessoren zu finden. In Tabelle 2 werden auch ihre Nachteile genannt – das soll nicht heißen, daß alle CPUs schlecht sind!

Herwig-Feichtinger

### Stichworte zum Inhalt:

8080, Z-80, 6800, 6502, 9980, SCMP, Vergleich



Bei der Aufzeichnung von Programmen auf Tonband und bei der Übertragung über eine Fernsprechleitung oder per Funk treten mit bestimmter Häufigkeit und Wahrscheinlichkeit Fehler auf, die zumindest erkannt, besser aber korrigiert werden müssen. Der folgende Beitrag beschreibt übliche Methoden hierzu.

# Vom Parity-Bit zur Kreuzparität

## Format von Datenübertragungen

Wenn hier von „Datenübertragungen“ die Rede ist, so ist damit auch die Aufzeichnung von Daten z. B. auf eine Kassette gemeint. In beiden Fällen geschieht die Codierung der Daten meist im ASCII-Format, wenn es sich um Text handelt, oder im 8-bit-binären Format, wenn ausschließlich Datenbytes übertragen werden sollen. Oft werden die Datenbytes aber auch in zwei „Nibbles“ aufgespalten und als zwei hexadezimale Ziffern im ASCII-Format übertragen.

ASCII heißt „American Standard Code for Information Interchange“ und repräsentiert ein beliebiges von max. 128 Schrift- und Steuerzeichen als 7-bit-Code. Eine ASCII-Tabelle findet sich z. B. in FUNKSCHAU 1979, Heft 7. Bei serieller Datenübertragung verwendet man meist ein Format nach Bild 1 (Start-Stop-Betrieb). Jedes Zeichen beginnt mit einem Startbit (log. 0). Dann folgen die sieben ASCII-Bits und ein Paritätsbit, das entweder konstant 0 oder 1 ist oder aber zur Fehlererkennung verwendet werden kann. Bei 110 Bd werden zwei Stopbits angehängt, sonst nur eines. Wird kein Zeichen ausgestrahlt, so entspricht die „Ruhelage“ der Stop-Polarität (log. 1). Zwischen den Zeichen sind beliebig lange Pausen zulässig (asynchrones Format).

## Fehlerhäufigkeit

Ein Maß für die Güte einer Datenübertragungs- oder -aufzeichnungseinrichtung ist die durchschnittliche Zahl der Fehler pro Bit (selbstverständlich immer kleiner als 1, theoretisch sogar kleiner als 0,5). Bei der Datenübertragung über eine öffentliche Fernsprechleitung kann man mit einer Bitfehlerwahrscheinlichkeit zwischen etwa  $10^{-4}$  und  $10^{-5}$  rechnen [1], je nach Leitungsqualität und Übertragungsgeschwindigkeit. Bei Tonbandaufzeichnungen lassen sich Werte bis etwa  $10^{-6}$  erreichen. Wenn mit einer Wahrscheinlichkeit von  $10^{-4}$  ein Bit falsch ist, so ist ein komplettes ASCII-Zeichen (10 bit mit Start- und Stopbits) bereits mit ei-

ner Wahrscheinlichkeit von  $10^{-3}$  falsch, d. h. durchschnittlich wird jedes tausendste Zeichen falsch übertragen. Geschieht die Übertragung von Datenbytes in Form ASCII-codierter Halbbytes (Nibbles), so ist sogar jedes 500. Byte falsch.

## Fehlerarten

Der am häufigsten auftretende Fehler ist, daß nur eines der insgesamt 10 oder 11 Bits eines ASCII-Zeichens (Start- und Stopbits mitgerechnet) z. B. durch einen kurzen Störimpuls falsch ist, d. h. invertiert ankommt. Ist das Startbit das falsche Bit, so wird es nicht als solches erkannt, und aufgrund der falschen Starterkennung sind die Wertigkeiten aller folgenden Bits verschoben, so daß u. U. alle Bits falsch ankommen. Trat der Fehler bei einem der folgenden Datenbits auf, so ist nur dieses falsch. Ein invertiert ankommendes Stopbit wird als (falsches) Startbit für das folgende Zeichen interpretiert.

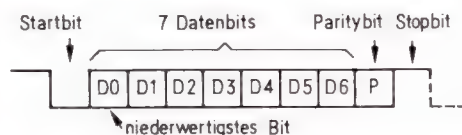


Bild 1. Serieller Übertragungsformat eines 7-bit-ASCII-Zeichens mit Paritätsbit

Eine bei Funkübertragungen häufig vorkommende Fehlerart wird durch sog. „Bündelstörungen“ hervorgerufen, die sich über mehrere Bits erstrecken, so daß innerhalb eines ASCII-Zeichens mehrere Bits falsch ankommen können.

Am Rande sei erwähnt, daß eine Resynchronisierung empfangsseitig schneller erfolgt, wenn nicht ein, sondern zwei Stopbits übertragen werden. Bei stark gestörten Übertragungen ist daher die Verwendung von zwei Stopbits empfehlenswert.

Bevor man Methoden zur Software-Fehlererkennung und -korrektur einsetzt, sollte man sich zunächst bemühen, die Hardware des Übertragungssystems optimal an das Datenformat an-

zupassen. Dazu gehören u. a. Amplitudenbegrenzer zum Ausgleich von Nf-Schwankungen und selektive Filter, die die wirksame Rausch- und Störbandbreite empfangsseitig verringern und nur das Frequenzspektrum des Datensignals ungeschwächt passieren lassen. Eine Verbesserung des Nf-Störabstands um nur 3 dB verringert die Bitfehlerhäufigkeit um rund eine Zehnerpotenz [1].

## Das Parity-Bit

Die einfachste Methode der Fehlererkennung ist das Parity-Bit (Paritätsbit). Es wird als letztes Bit eines Zeichens als Prüfsumme aller vorhergehenden Bits ausgestrahlt. (Von der Prüfsumme wird natürlich nur das niederwertigste Bit als Parity verwendet.) Manche Mikroprozessoren besitzen spezielle Befehle zum Errechnen der Parity (8080: CPE, JPE, JPO), bei anderen ist ein kleines Programm dazu erforderlich, das beim 6502 etwa so aussehen kann:

```
0000 LDX # 08
0002 LDA # 00
0004 ROR FE
0006 ADC # 00
0008 DEX
0009 BNE 0004
```

Das Byte, dessen Parity errechnet werden soll, steht dabei in der Zelle 00FE. Nach der Routine enthält der Akku die komplette Prüfsumme über alle Bits und das Carry-Flag im Statusregister das Parity-Bit. (Die meisten UART-Bausteine gewinnen das Paritätsbit per Hardware selbst.)

Beim Empfang eines ASCII-Zeichens wird das Parity-Bit neu errechnet und mit dem ausgestrahlten verglichen. Trat nur ein Bitfehler innerhalb des Zeichens auf, so kann dieser mit dieser Methode sicher erkannt werden. Wurden aber zwei Bits falsch empfangen, so ist das Parity-Bit zur Fehlererkennung nicht geeignet, da dann trotz der Fehler die Parität stimmt.

Die Sicherheit, mit der durch Paritätsprüfung Bitfehler erkannt werden, ist deshalb davon abhängig, wie häufig Mehrbitfehler innerhalb eines Zeichens



auftreten und damit wiederum von der Qualität des Mediums: Je geringer die Bitfehlerwahrscheinlichkeit ist, desto besser funktioniert die Fehlererkennung durch Paritätsbildung. Unter bestimmten Voraussetzungen wird ein fehlerhaftes Zeichen nur mit 50% Wahrscheinlichkeit als solches erkannt.

Übrigens unterscheidet man zwischen gerader (even) und ungerader (odd) Parität; letztere entsteht durch Invertieren der Bit-Prüfsumme. Beide Paritätsarten finden bei der Datenübertragung Anwendung.

## Prüfsummen

Eine wesentlich sicherere Methode zur Fehlererkennung, z. B. bei der Aufzeichnung von Mikrocomputer-Programmen auf Tonband-Kassetten, ist die Prüfsummenbildung. Hierbei werden alle Bytes eines Datenblockes (er kann z. B. aus 64, 80 oder 256 Zeichen bestehen) einfach aufsummiert, wobei ein eventuell entstehender Übertrag (Wertigkeit  $2^8$ ) nicht berücksichtigt wird. Das so entstehende Prüfsummen-Byte wird im Anschluß an den Datenblock übertragen. Häufig arbeitet man auch mit einer 16-bit-Prüfsumme, für die dann zwei Bytes benötigt werden. Die Wahrscheinlichkeit, daß sich bei der Übertragung die Fehler im Datenblock und in der Prüfsumme gerade aufheben, ist absolut vernachlässigbar, so daß diese Fehlererkennung sehr sicher funktioniert. Sie wird deshalb sowohl bei Kassetten als auch bei Lochstreifen praktisch ausschließlich verwendet. Ein übliches Format ist folgendes:

```
;180000000102030405060708090A0B
0C0D0E0F1011121314151617012C
Der Strichpunkt kündigt den Anfang
der Übertragung an. 18 ist die hexade-
zimale Anzahl der folgenden Datenby-
tes, 0000 die Anfangsadresse. Dann fol-
gen hex 18 Bytes (hier 00...17) und de-
ren 16-bit-Prüfsumme (012C). Manch-
mal wird noch zusätzlich die Anzahl al-
ler Datenblöcke aufgezeichnet, um kon-
trollieren zu können, ob wirklich alle
empfangen wurden. In obigem Beispiel
würde dann noch angehängt:
;0000010001
```

Die zwei ersten Nullen zeigen an, daß keine Daten mehr folgen; dann folgt die vierstellige (hexadezimale) Anzahl der übertragenen Blöcke und, nochmals vierstellig, deren Prüfsumme, die natürlich hier identisch mit der Blockzahl ist.

Eine Abart der Prüfsummen-Bildung ist die CRC-Methode (Cyclic Redundancy Check). Hierbei wird aus allen Datenbytes eines Blocks eine 16-bit-Zahl errechnet, die in Form von zwei CRC-Bytes mit aufgezeichnet wird. Beim Lesen muß die gesamte empfangene Bitfolge, wenn man sie als vielstel-

lige Binärzahl betrachtet, durch das 16-bit-CRC-Wort teilbar sein. Die CRC-Methode ist vor allem bei Disketten-Laufwerken üblich [2].

## Der Hamming-Code

Die Parity-Bits und Prüfsummen sind zwar geeignet, Fehler zu beheben und damit den Benutzer der Übertragungseinrichtung oder des Kassettenrecorders aufzufordern, das ganze Spiel zu wiederholen; sie können aber die erkannten Fehler nicht selbständig korrigieren, da ihre Redundanz, d. h. zusätzliche Information, dafür zu gering ist.

Eine Methode zur Fehlerkorrektur wäre z. B., jedes Byte mehrmals auszustrahlen und nur diejenigen Bytes auszuwerten, deren Parity stimmt. Dieser Weg ist aber verschwenderisch, denn es wird dadurch ein Vielfaches an Übertragungszeit benötigt.

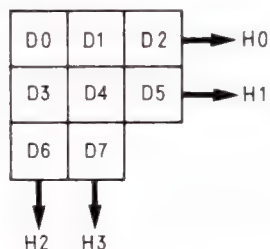


Bild 2. Die Position des falsch übertragenen Datenbits (D 0...D 7) wird durch die Bits H 0...H 3 ermittelt

Der Mathematiker Hamming fand einen Weg, wie sich ein 8-bit-Datenwort durch Hinzufügen von nur vier Bits (den Hamming-Bits) beim Empfang korrigieren läßt, wenn nur 1 Bit fehlerhaft übertragen wurde. Bild 2 zeigt, wie das funktioniert: Das fehlerhafte Bit läßt sich durch eine aus  $2 \times 2$  Hamming-Bits gebildete Matrix lokalisieren und durch Inversion korrigieren. Errechnet z. B. der Empfänger ein mit dem gesendeten nicht identisches Hamming-Bit H 3, sind die anderen aber gleich, so wurde D 7 falsch übertragen und braucht jetzt für eine Korrektur nur invertiert zu werden. Stimmen dagegen H 0 und H 3 nicht mit der Aufzeichnung überein, so muß D 1 invertiert werden.

Selbstverständlich kann aber auch bei der Übertragung eines Hamming-Bits ein Fehler aufgetreten sein. Um dies zu erkennen, verwendet man auch hier sicherheitshalber ein Paritätsbit, das z. B. statt D 7 ausgestrahlt wird (ein ASCII-Zeichen belegt ja nur D 0...D 6). Wenn man annimmt, daß pro Zeichen (inklusive der Hamming-Bits, also für 12 Bits) nur ein Bitfehler auftritt, so wird er korrigiert, wenn er – bei falschem Parity-Bit – innerhalb D 0...D 6 lag. Ist das Parity-Bit richtig, so kann ein Fehler höchstens in den Hamming-Bits aufgetreten sein, und es wird keine Korrektur der Datenbits vorgenommen.

## Die Kreuzparität

Verwendet man Datenblöcke konstanter Länge, z. B. acht Bytes pro Block, so kann das Hamming-Prinzip auf diese Blöcke erweitert werden (Bild 3). Es ergeben sich jetzt 16 redundante Bits, die wiederum eine Fehlerkorrektur ermöglichen. Die Redundanz ist relativ gering: Für acht Datenbytes werden nur zwei Paritätsbytes zur Fehlerkorrektur benötigt (25% Redundanz). Dafür darf hier immerhin über zehn Bytes hinweg nur ein Bitfehler auftreten, damit das Verfahren funktioniert. Zusammen mit der Tatsache, daß die Kreuzparität nur bei festen Datenblock-Formaten anwendbar ist, ist dies der Grund, warum sie keine so große Verbreitung wie der Hamming-Code fand.

Herwig Feichtinger

## Literatur

- [1] Bacher, Grunow, Schierenbeck: Datenübertragung, Siemens, München 1978, ISBN 3-8009-1270-8.
- [2] Lesea, A.; Zaks, R.: Mikroprozessor-Interface-Techniken, Micro-Shop Bodensee, ISBN 3-922187-00-5.

## Stichworte zum Inhalt

Datenformat, ASCII, Parität, Parity, Prüfsumme, Hamming-Code, Bitfehler, Kreuzparität, Fehlererkennung, Fehlerkorrektur.

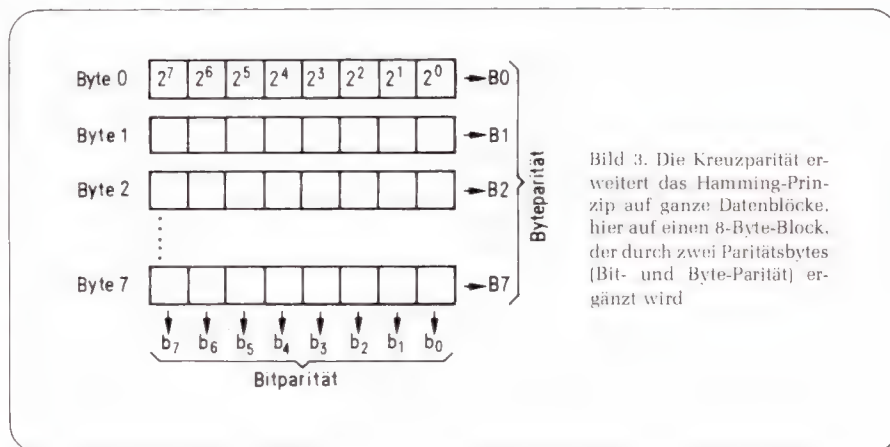


Bild 3. Die Kreuzparität erweitert das Hamming-Prinzip auf ganze Datenblöcke, hier auf einen 8-Byte-Block, der durch zwei Paritätsbytes (Bit- und Byte-Parität) ergänzt wird



# Zahlendarstellung im PET

In FUNKSCHAU 1979, Heft 9, wurde in „Gute Noten dank PET“ von Hans-Georg Joepgen das Problem aufgeworfen, daß beim PET bei komprimierter Programmierung ein anderes Ergebnis auftritt als bei getrennter Programmierung. Die beiden Programmteile lauten:

Fassung 1:

10 S = 66/20

20 E = INT (10 \* S)/10

Ergebnis: E = 3,2

Fassung 2:

40 S = 66/20

50 E = S \* 10 : E = INT(E) : E = E/10

E = 3,3

Im Anschluß daran wird nach der Ursache dieses Verhaltens gefragt, wobei anscheinend auch Commodore Deutschland keine Lösung wußte. Die Lösung dieses Problems liegt nicht darin, daß der PET ein Herz für schwache Schüler hat, sondern es handelt sich um ein Rundungsphänomen. Um das genauer erläutern zu können, muß näher auf die Zahlendarstellung im PET, die verwendete Rundung, die Multiplikation bzw. auch Division und die Funktion INT eingegangen werden.

## Zahlendarstellung

Bevor auf die Zahlendarstellung im PET eingegangen werden kann, muß man sich kurz die Theorie der Konvertierung von Zahlendarstellungen vergegenwärtigen.

### Konvertierung von natürlichen Zahlen

In einem Zahlensystem mit der Basis B wird eine natürliche Zahl n wie folgt dargestellt:

$$n = \sum_{i=0}^N b_i \cdot B^i =$$

$$b_N \cdot B^N + b_{N-1} \cdot B^{N-1} + \dots$$

$$+ b_1 \cdot B + b_0 \cdot B^0 \quad (1)$$

Mit Hilfe des Horner-Schemas läßt sie sich aber auch folgendermaßen ansprechen:

$$n = (\dots(((b_N \cdot B + b_{N-1}) \cdot B + b_{N-2}) \cdot B + b_{N-3}) \cdot B + \dots + b_1) \cdot B + b_0 \cdot 1 \quad (2)$$

Beispiel 1: Dezimalsystem, B = 10

$$n = 1234 =$$

$$1 \cdot 10^3 + 2 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0 =$$

$$= b_3 \cdot B^3 + b_2 \cdot B^2 + b_1 \cdot B^1 + b_0 \cdot B^0$$

oder

$$n = ((1 \cdot 10 + 2) \cdot 10 + 3) \cdot 10 + 4 \cdot 1 =$$

$$((b_3 \cdot B + b_2) \cdot B + b_1) \cdot B + b_0 \cdot 1$$

Aus Gleichung (2) sieht man, daß man die einzelnen Ziffern  $b_0, b_1, \dots, b_N$  dadurch erhält, daß man die Zahl n durch B dividiert, nach der Division den Rest absondert und notiert. Der Quotient wird dann wieder durch B dividiert usw. bis ein Quotient 0 entsteht. Wenn man nun die einzelnen Reste nebeneinander schreibt, erhält man genau die Zifferndarstellung der Zahl n im System mit der Basis B.

Die Konvertierung geht also nach folgendem Schema vor sich:

$$n : B = q_0 + r_0/B$$

$$q_0 : B = q_1 + r_1/B$$

$$q_1 : B = r_2 + r_2/B$$

$$q_{N-1} : B = 0 + r_N/B$$

wobei Rest  $r_i = i$ -te Ziffer  $b_i$   
( $i = 0, 1, \dots, N$ )

Beispiel 2: Konvertierung der Zahl  $123_{10}$  ins Dualsystem

$$123 : 2 = 61, \text{ Rest } 1; \quad b_0 = 1.$$

$$61 : 2 = 30, \text{ Rest } 1; \quad b_1 = 1.$$

$$30 : 2 = 15, \text{ Rest } 0; \quad b_2 = 0.$$

$$15 : 2 = 7, \text{ Rest } 1; \quad b_3 = 1.$$

$$7 : 2 = 3, \text{ Rest } 1; \quad b_4 = 1.$$

$$3 : 2 = 1, \text{ Rest } 1; \quad b_5 = 1.$$

$$1 : 2 = 0, \text{ Rest } 1; \quad b_6 = 1.$$

$$\text{D.h. } 123_{10} = 1111011_2.$$

Probe:

$$64 + 32 + 16 + 8 + 2 + 2 + 1 = 123.$$

Beispiel 3: Konvertierung der Zahl  $1111011_2$  ins Dezimalsystem

$$1111011 : 1010 =$$

$$1100 \text{ Rest } 11 \quad b_0 = 3_{10}$$

$$1100 : 1010 = 1 \text{ Rest } 10 \quad b_1 = 2_{10}$$

$$1 : 1010 = 0 \text{ Rest } 1 \quad b_2 = 1_{10}$$

$$\text{d.h. } 1111011_2 = 123_{10}$$

Konvertierung von echt gebrochenen Zahlen

Eine echt gebrochene Zahl y

$$(0 < |y| < 1)$$

$$y = \sum_{i=-1}^{-M} b_i \cdot B^i = b_{-1} \cdot B^{-1} + \dots + b_{-M} \cdot B^{-M} \quad (3)$$

kann entsprechend mit dem Horner-Schema dargestellt werden:

$$y = \frac{1}{B} (b_{-1} + \frac{1}{B} (b_{-2} + \dots + \frac{1}{B} (b_{-M+1} + \frac{1}{B} b_{-M} \dots))) \quad (4)$$

Beispiel 4: Dezimalsystem, B = 10

$$y = 0,1234 = 1 \cdot 10^{-1} + 2 \cdot 10^{-2} + 3 \cdot 10^{-3} + 4 \cdot 10^{-4} =$$

$$= b_{-1} \cdot B^{-1} + b_{-2} \cdot B^{-2} + b_{-3} \cdot B^{-3} + b_{-4} \cdot B^{-4} \text{ oder}$$

$$y = \frac{1}{10} (1 + \frac{1}{10} (2 + \frac{1}{10} (3 + \frac{1}{10} 4)))$$

Sinngemäß zur Methode nach Gleichung (2) erfolgt hier nach Gleichung (4) die Konvertierung durch sukzessive Division durch bzw. Multiplikation mit B.

Beispiel 5: Konvertierung der Zahl  $0,125_{10}$  ins Dualsystem

$$0,125 \cdot 2 = 0,25; \quad \text{Überlauf } \ddot{u}_{-1} = 0,$$

$$0,25 \cdot 2 = 0,5; \quad \text{Überlauf } \ddot{u}_{-2} = 0,$$

$$0,5 \cdot 2 = 1,0; \quad \text{Überlauf } \ddot{u}_{-3} = 1,$$

wobei  $\ddot{u}_i = i$ -te Ziffer  $b_i$

wobei ( $i = -1, -2, \dots, -M$ )

$$\text{d.h. } 0,125_{10} = 0,001_2$$

$$\text{Probe: } 2^{-3} = \frac{1}{8} = 0,125$$

Da es sich bei  $0,125_{10}$  um eine Zahl handelt, die eine Potenz von 2 ist, ergibt sich nach einigen Multiplikationen die Zahl 0 hinter dem Komma, womit die sukzessive Multiplikation abgeschlossen ist.



**Beispiel 6:** Konvertierung der Zahl  $0,13_{10}$  ins Dualsystem

$0,13 \cdot 2 = 0,26$ ; Überlauf  $\ddot{u}_{-1} = 0$ ;  
 $0,26 \cdot 2 = 0,52$ ; Überlauf  $\ddot{u}_{-2} = 0$ ;  
 $0,52 \cdot 2 = 1,04$ ; Überlauf  $\ddot{u}_{-3} = 1$ ;  
 $0,04 \cdot 2 = 0,08$ ; Überlauf  $\ddot{u}_{-4} = 0$ ;  
 $0,08 \cdot 2 = 0,16$ ; Überlauf  $\ddot{u}_{-5} = 0$ ;

d.h.  $0,13_{10} \approx 0,00100_2$

Probe:  $2^{-3} = \frac{1}{8} = 0,125 \approx 0,13$

Die Genauigkeit der Konvertierung hängt nun, da sich nie 0 hinter dem Komma ergibt, davon ab, bei welchem M die Konvertierung abbricht, mit wievielen Stellen hinter dem Komma man sich also begnügt.

**Beispiel 7:** Konvertierung der Zahl  $0,001_2$  ins Dezimalsystem

$0,001 \cdot 1010 = 1,010$ ; Überlauf 1;  
 $b_{-1} = 1_{10}$ .  
 $0,01 \cdot 1010 = 10,10$ ; Überlauf 10;  
 $b_{-2} = 2_{10}$ .  
 $0,1 \cdot 1010 = 101,0$ ; Überlauf 101;  
 $b_{-3} = 5_{10}$ .  
d.h.:  $0,001_2 = 0,125_{10}$

**Konvertierung von gebrochenen Zahlen**

Eine gebrochene Zahl  $x$

$$x = \sum_{i=-M}^N b_i \cdot B^i = \sum_{i=0}^N b_i \cdot B^i + \sum_{i=-M}^{-1} b_i \cdot B^i$$

läßt sich auf zwei Arten konvertieren, einmal durch getrennte Konvertierung des ganzen und des echt gebrochenen Anteils oder durch eine Normierung, die  $x$  in eine ganze Zahl oder in eine echt gebrochene Zahl überführt und anschließende Konvertierung durch sukzessive Division bzw. Multiplikation mit Rest bzw. Überlauf.

Die Normierung auf eine echt gebrochene Zahl würde folgendes Ergebnis liefern:

$$z = \pm p : B^q \quad 0 < p < 1$$

$q$  = Maßstabsfaktor, z. B.  $0 \leq q \leq 99$  im Dezimalsystem.

Für eine richtige Gleitkommadarstellung ist es aber notwendig, daß in  $p$  auf das Komma sofort eine Ziffer 0 folgt, es müssen also auch negative Exponenten von  $B$  zugelassen werden. Dazu kann entweder eine zusätzliche Exponentenvorzeichenstelle verwendet werden, es kann aber auch eine sogenannte Charakteristik  $q'$  eingeführt werden.

$q' = q + q_0$   $q_0$  = Konstante  
z. B.  $q_0 = 50$ : für  $q = -50$  bis  $+49$  ergibt sich  $q' = 0$  bis  $99$

Die Darstellung einer Gleitkommazahl nur durch  $p$  und  $q'$  heißt halblogarithmische Darstellung, wobei  $p$  als Mantisse bezeichnet wird.

**Zahldarstellung im PET**

Beim PET werden die Zahlen nun so normiert, daß das Komma nach der ersten Ziffer, also nach dem vordersten Einser steht. Dazu wird solange durch 2 dividiert bzw. mit 2 multipliziert, bis das Ergebnis eine Zahl  $p$  mit  $1 \leq p < 2$  ist. Die Anzahl der Divisionen bzw. Multiplikationen gibt als positiver bzw. negativer Zweierkomponent den Maßstabsfaktor, also die Wertigkeit des Einsers vor dem Komma, an. Die Zahl hinter dem Komma, die eine echt gebrochene Zahl ist, wird dann nach Gleichung (4) (Beispiele 5 und 6) konvertiert und ergibt die Binärziffern hinter dem Komma.

**Beispiel 8:** Konvertierung von Dezimalzahlen in PET-Dualzahlen

$5 : 2 = 2,5$   
 $2,5 : 2 = 1,25$  2 Divisionen  
 $\rightarrow 1, \dots \cdot 2^2$

$0,25 \cdot 2 = 0,5$  Überlauf 0  
 $0,5 \cdot 2 = 1,0$  Überlauf 1  
d.h.  $5_{10} = 1,01 \cdot 2^2$

Probe:  $2^2 + 2^0 = 5$

$3,3 : 2 = 1,65$  1 Division  $\rightarrow 1, \dots \cdot 2^1$   
 $0,65 \cdot 2 = 1,3$  Überlauf 2  
 $0,3 \cdot 2 = 0,6$  Überlauf 0  
 $0,6 \cdot 2 = 1,2$  Überlauf 1  
 $0,2 \cdot 2 = 0,4$  Überlauf 0  
 $0,4 \cdot 2 = 0,8$  Überlauf 0  
 $0,8 \cdot 2 = 1,6$  Überlauf 1  
 $0,6 \cdot 2 = 1,2$  Überlauf 1

Da wieder 0,6 zur Multiplikation mit 2 kommt, tritt eine Periode auf, d.h.  $3,3_{10} = 1,101001 \cdot 2^1$

Probe:  $2^1 + 2^0 + 2^{-2} + 2^{-5} = 2 + 1 + 0,25 + 0,03125 = 3,28125 \approx 3,3$   
 $0,7 \cdot 2 = 1,4$ ; 1 Multiplikation  $\rightarrow 1, \dots \cdot 2^{-1}$

$0,4 \cdot 2 = 0,8$ ; Überlauf 0;  
 $0,8 \cdot 2 = 1,6$ ; Überlauf 1;  
 $0,6 \cdot 2 = 1,2$ ; Überlauf 1;  
 $0,2 \cdot 2 = 0,4$ ; Überlauf 0;  
 $0,4 \cdot 2 = 0,8$ ; Überlauf 0;

⋮  
⋮

d.h.  $0,7_{10} = 1,0110 \cdot 2^{-1}$

Probe:  $2^{-1} + 2^{-3} + 2^{-4} = 0,5 + 0,125 + 0,0625 = 0,6875 \approx 0,7$

$0,07 \cdot 2 = 0,14$ ;

$0,14 \cdot 2 = 0,28$ ;

$0,28 \cdot 2 = 0,56$ ;

$0,56 \cdot 2 = 1,12$ ; 4 Multiplikationen  
 $\rightarrow 1, \dots \cdot 2^{-4}$

$0,12 \cdot 2 = 0,24$ ; Überlauf 0;

$0,24 \cdot 2 = 0,48$ ; Überlauf 0;

$0,48 \cdot 2 = 0,96$ ; Überlauf 0;

$0,96 \cdot 2 = 1,92$ ; Überlauf 1;

$0,92 \cdot 2 = 1,84$ ; Überlauf 1;

$0,84 \cdot 2 = 1,68$ ; Überlauf 1;

$0,68 \cdot 2 = 1,36$ ; Überlauf 1;

$0,36 \cdot 2 = 0,72$ ; Überlauf 0;

⋮  
⋮

d.h.  $0,07_{10} \approx 1,00011110 \cdot 2^{-4}$

Probe:  $2^{-4} + 2^{-8} + 2^{-9} + 2^{-10} + 2^{-11} =$   
 $= 0,0625 + 0,00390625 + 0,001953125$   
 $+ 0,0009765625 +$   
 $+ 0,00048828125 = 0,06982821875 \approx 0,07$

Der PET verwendet zur Zahldarstellung 5 Byte, wovon eines zur Speicherung der Charakteristik, die anderen 4 für die Mantisse reserviert sind. Das höchstwertige Bit ist das Vorzeichenbit, so daß noch 31 bit für die Mantisse bleiben, wobei nur die Ziffern nach dem Komma abgespeichert werden. Ist die Zahl positiv, ist das Vorzeichenbit 0, ist sie negativ, so ist es 1, Mantisse und Exponent bleiben gleich.

Die zur Bildung der Charakteristik notwendige Konstante ist  $129_{10}$  ( $81_{16}$ ). Positive Exponenten werden also zu  $81_{16}$  addiert, negative subtrahiert. Damit kann der größte Zweierexponent nur  $126_{10}$  ( $255_{16}$ ) betragen, was als größtmögliche Zahl  $\approx 1,7 \cdot 10^{38}$  ( $2^{126} \approx 8,5 \cdot 10^{37}$ ) erlaubt. Die kleinste darstellbare Zahl hat demnach den Zweierexponenten  $-129$  und beträgt  $\approx 1,5 \cdot 10^{-39}$ . Zahlen  $< 3 \cdot 10^{-39}$  werden aber als 0 interpretiert.

Die in Beispiel 8 gegebenen Zahlen lauten also in PET-Darstellung:

$5_{10} = \underline{83} \quad \underline{20} \quad \underline{00} \quad \underline{00} \quad \underline{00}$

Exp. 0 0100000... 0

positiv ↑ in hexadezimaler  
↑ Darstellung der  
Byteinhalte

Überläufe der Multiplikationen  
(dezimal 131 32 0 0 0)

$3,3_{10} = 82 \ 53 \ 33 \ 33 \ 33_{16}$   
(130 83 51 51 51 dezimal)

$0,7_{10} = 80 \ 33 \ 33 \ 33 \ 33_{16}$   
(128 51 51 51 51 dezimal)

$0,06_{10} = 7D \ 0F \ 5C \ 28 \ F6_{16}$   
(125 15 92 40 246 dezimal)

Ein gründliches Studium der Speichereinhalte der Page 0 (Adressen  $0 \dots 255_{10}$ ), über deren Belegung durch das Betriebssystem leider keine Unterlagen erhältlich sind, zeigt, daß in den Speicherzellen 124, 125 ein Zeiger auf den Anfang der Variablen-tabelle steht, und zwar in der für den  $\mu P$  6502 üblichen Form: In 124 das niederwertige Byte, in 125 das höherwertige. Rechnet man diese Hexadezimalzahl in eine Dezimalzahl um und holt sich durch einen PEEK-Befehl den Inhalt dieser Speicherzelle, so sieht man, daß in ihr der ASCII-Code des ersten Buchstabens der ersten vorkommenden Variablen steht.

Da das PET-BASIC als normale Variablennamen höchstens zwei Buchstaben bzw. Buchstabe + Ziffer zuläßt, sind zur Speicherung des Namens zwei



Bytes erforderlich. Auf diese 2 Bytes folgen dann die 5 Bytes der Zahl. Handelt es sich um die Zahl 0, so sind diese 5 Bytes gleich 0.

Beispiel 9:

10 AB = - 5  
20 F = 3.3

Das bedeutet, daß nach Abarbeitung dieser Zeilen folgende Bytes abgespeichert sind:

Adresse	Inhalt	
124	18 <sub>16</sub> (24 <sub>10</sub> )	} → 4 × 256 + 24 = 1048
125	4	
1048	41 42 83 A0 00 00 00 <sub>16</sub>	
	A B	- 5
		(65 66 131 160 0 0 0 dez.)
1055	46 00 82 53 33 33 33 <sub>16</sub>	
	F	+ 3.3
		(70 0 130 83 51 51 51 dez.)

Für die Speicherung der außerdem noch möglichen Intervariablen (Bereich: -32768 bis 32767), die durch ein %-Zeichen nach dem Namen gekennzeichnet werden, werden, obwohl die Zahlen selbst umgerechnet in Hexadezimalzahlen in Zweierkomplementdar-

stellung in 2 Bytes abgespeichert werden, trotzdem auch 7 Bytes verwendet. In den ersten beiden Bytes wird wieder der Name abgespeichert, wobei als Kennzeichnung der Intervariablen zum ASCII-Code der Buchstaben bzw. Ziffern noch 80<sub>16</sub> (128<sub>10</sub>) addiert wird, in den nächsten beiden Bytes die Zahl, die restlichen drei sind 0, wodurch also keine Platzersparnis gegenüber den normalen Variablen eintritt! (Eine Platzersparnis tritt nur dann auf, wenn man Felder von Intervariablen benützt.)

Beispiel 10:

10 FF% = 12345 (= 3039<sub>16</sub>)  
20 G% = -5 (= FFFB<sub>16</sub>)

Adresse	Inhalt	
124	28 <sub>10</sub>	} 4 · 256 + 28 = 1052
125	4	
1052	C6 C6 30 39 00 00 00 <sub>16</sub>	
	FF + 80 <sub>16</sub>	Zahl
		(198 198 48 57 0 0 0 dez.)
1059	C7 80 FF FB 00 00 00 <sub>16</sub>	
	G + 80 <sub>16</sub>	
		(199 128 255 251 0 0 0 dez.)

geschlossen wird, also durch =, +, -, x, /, . Auch die Funktionen SQR, LOG, EXP, SIN, COS, TAN, ATN, ABS runden zuerst das Ergebnis der Multiplikation, bevor die Funktion ausgeführt wird.

Beispiel 12: 10 A = 3,3 · 10

Durch das =-Zeichen wird die Rundung veranlaßt, so daß A folgende Dualzahl ist:

1.00001000000000000000000000000000  
· 2<sup>5</sup> = 33<sub>10</sub>  
bzw. 86 04 00 00 00<sub>16</sub> (134 4 0 0 0 dez.)

## Division

Bei der Division gilt das Analoge zur Multiplikation. Der Quotient wird auch über eine 31stellige Mantisse hinaus genauer berechnet und dann durch einen der Operatoren =, +, -, x, /, und bei den oben genannten Funktionen gerundet.

Beispiel 13: S = 66/20

66<sub>10</sub> = 1.00001 · 2<sup>6</sup>  
20<sub>10</sub> = 1.01 · 2<sup>4</sup>  
1.00001 · 2<sup>6</sup> : 1.01 · 2<sup>4</sup> = 0.11010011 · 2<sup>2</sup>  
= 1.1010011 · 2<sup>1</sup>

## Multiplikation

Zwei Zahlen bzw. Variablen werden nun so miteinander multipliziert, daß man zuerst die binären Mantissen miteinander multipliziert. Ergibt sich bei dieser Multiplikation mehr als eine Stelle vor dem Komma, so wird das Komma wieder so weit nach links geschoben, bis nur mehr eine Stelle vor dem Komma steht. Die Anzahl der Verschiebungen ergibt zusammen mit den Zweierexponenten der ursprünglichen Zahlen den Exponenten des Ergebnisses.

Beispiel 11: 3,3 : 10 (es handelt sich dabei um die Multiplikation des Problems von FUNKSCHAU 9/1979)  
3,3<sub>10</sub> = 82 53 33 33 33<sub>16</sub>  
10<sub>10</sub> = 84 20 00 00 00<sub>16</sub>

1.1010011001100110011001100110011 : 2<sup>1</sup> · 1.010 ... 0.2<sup>3</sup>  
1101001100110011001100110011 29 Nullen  
10.0000111111111111111111111111110 ... 0.2<sup>4</sup> =  
29 Nullen  
= 1.0000011111111111111111111111111 | 1111 · 2<sup>5</sup>  
31 Binärstellen

Das Ergebnis der Multiplikation ist natürlich länger als die beiden Faktoren. Es muß daher auf 31 Stellen hinter dem Komma begrenzt werden. Dabei wird aber nicht einfach nach der 31. Stelle abgeschnitten, sondern gerundet.

## Rundung

Die Rundung tritt auch schon bei der Konvertierung der Dezimalzahlen in die Dualzahlen auf. Hier wird, soweit sich das hier bisher feststellen ließ, mit mindestens 3 bit weitergerechnet und dann aufgerundet, wenn der Zahlenwert dieses Halbbytes > 7 ist. Bei der Multiplikation wird die Rundung erst dann durchgeführt, wenn die Operation durch einen weiteren Operator ab-

- 101
110
- 101
11
- 0
110
- 101
10
0
- 100
0
1000
- 101
110
- 101
1
.
.
.

In diesem Fall wird nicht aufgerundet, da das nach der 31. Stelle nach dem Komma folgende Halbbyte den Wert 3 hat.

S = 1.1010011 · 2<sup>1</sup> = 82 53 33 33 33<sub>16</sub>  
(130 83 51 51 51 dez.)



## Die Funktion INT

Die Funktion INT bildet das kleinste Ganze einer Zahl, liefert als Ergebnis also die nächst kleinere ganze Zahl. Für positive Zahlen bedeutet das, daß das Komma der Mantisse entsprechend dem Zweierexponenten verschoben wird und daß alle Bits hinter dem Komma auf 0 gesetzt werden. Das Komma wird anschließend, wenn die entstandene Zahl nicht Null ist, wieder solange nach links verschoben, bis nur mehr eine Ziffer vor dem Komma steht. Die Anzahl der Verschiebungen ist dann wieder gleich dem Exponenten.

Beispiel 14:  $\text{INT}(3.3) = 3$ ,  $\text{INT}(0.5) = 0$

$3.3_{10} = 1.1010011001100$   
 $110011001100110011 \cdot 2^1 =$   
 $= 11.010011001100$   
 $1100110011001100110011$   
 $\text{INT}(3.3) = 11.00...0 =$   
 $1.100...0 \cdot 2^1 = 3_{10}$   
 $0.5_{10} = 1.00...0 \cdot 2^{-1} = 0.100...0$   
 $\text{INT}(0.5) = 0.00...0 = 0_{10}$

Bei negativen Zahlen wird auch das Komma entsprechend dem Zweierexponenten verschoben. Stehen dann hinter dem Komma nur Nullen, so ist das schon die gewünschte ganze Zahl. Andernfalls werden die Zahl vor dem Komma um 1 erhöht, nach dem Komma alle Stellen auf 0 gesetzt und danach das Komma wieder nach links verschoben, bis nur mehr eine Ziffer vor dem Komma steht.

Beispiel 15:  $\text{INT}(-0.99) = -1$ ,  
 $\text{INT}(-5.3) = -6$

$-0.99_{10} = 80 \text{ FD } 70 \text{ A3 D6}_{16}$   
 $= -1.111110101110000$   
 $1010001111010110 \cdot 2^{-1}$   
 $= -0.111111010111000$   
 $01010001111010110$

$\text{INT}(-0.99) = -1.00...0$   
 $(= 81 \text{ 80 } 00 \text{ 00 } 00_{16}) = -1_{10}$   
 $-5.3_{10} = 83 \text{ A9 } 99 \text{ 99 } 9\text{A}_{16} =$   
 $= -1.010100110011001$   
 $1001100110011010 \cdot 2^2 =$   
 $= -101.010011001100110$   
 $01100110011010$

$\text{INT}(-5.3) = -110.00...0 =$   
 $-1.100...0 \cdot 2^2 =$   
 $(= 83 \text{ C0 } 00 \text{ 00 } 00_{16}) = -6_{10}$

## Die Lösung des Problems...

Wie schon gesagt wurde, wird die Rundung erst durchgeführt, wenn die Operation durch einen weiteren Operator abgeschlossen wird. Die Funktion INT erfüllt diese Bedingungen nicht, es wird also nicht gerundet.

Das Ergebnis der Multiplikation wird von Beispiel 11 übernommen:

$S \cdot 10 = 3.3_{10} \cdot 10_{10} =$   
 $1.000001111111111111111111111111$   
 $1111 \cdot 2^5 =$   
 $= 100000.111111111111111111111111$   
 $111111$

Fassung 1:

$10 \text{ S} = 66/20$   
 $20 \text{ E} = \text{INT}(10 \cdot \text{S})/10$

$\text{INT}(S \cdot 10) = 100000.0... =$   
 $1.00...0 \cdot 2^5 = 32_{10}$   
 $\text{INT}(S \cdot 10)/10 = 1.10011001100$   
 $11001100110011001100/1100.2^1$   
 $\text{E} = 1.10011001100110011001100$   
 $11001101 \cdot 2^1 = 82 \text{ 4C CC CC}$   
 $\text{CC CD}_{16} =$   
 $= (130 \text{ 76 } 204 \text{ 204 } 205 \text{ dez.})$   
 $= 3.2_{10}$

Fassung 2:

$10 \text{ S} = 66/20$   
 $20 \text{ E} = S \cdot 10 : \text{E} = \text{INT}(\text{E}) : \text{E} = \text{E}/10$   
 $\text{E} = S \cdot 10 = 1.0000100...0$   
 $0 \cdot 2^5 = 10001.00...0$   
 $\text{INT}(\text{E}) = 10001.00...0$   
 $\text{E} = 1.00001 \cdot 2^5 = 33_{10}$   
 $\text{E}/10 = 1.10100110011001100$   
 $11001100110011/0011 \cdot 2^1$

$\text{E} = 1.101001100110011001100$   
 $1100110011 \cdot 2^1 =$   
 $82 \text{ 53 } 33 \text{ 33 } 33_{16} =$   
 $= (130 \text{ 83 } 51 \text{ 51 } 51 \text{ dez.}) = 3.3_{10}$

Man sieht also, daß die Differenz der Ergebnisse nach Fassung 1 und 2 nur dadurch zustande kommt, daß durch die Funktion INT die Rundung nicht durchgeführt wird. Will man das Abschneiden vom Beispiel in Heft 9/79 trotzdem in einem Befehl durchführen, so muß durch Multiplikation oder Division von 1 vor Anwendung der INT-Funktion dafür gesorgt werden, daß gerundet wird:

$10 \text{ S} = 66/20$   
 $20 \text{ E} = \text{INT}(10 \cdot \text{S} \cdot 1)/10$

Ergebnis:  $\text{E} = 3.3$

## Literatur

Claus, V.: Einführung in die Informatik, Teubner, 1975  
Klar, R.: Digitale Rechenautomaten, Göschen, 1976  
Dworatschek, S.: Einführung in die Datenverarbeitung, De Gruyter, 1971

## Stichworte zum Inhalt

Zahldarstellung, Zahlumwandlung, Konvertierung, Rundung, Multiplikation, Division, INT.

# Anwendungsbeispiele für den Mikroprozessor 6502

Die nach dem legendären „First Book of KIM“ wohl umfassendste Programmsammlung für die CPU 6502 erschien jetzt in der Radio-Praktiker-Bücherei des Franzis-Verlages (RPB 173). Ein einleitendes Kapitel befaßt sich mit den heute erhältlichen 6502-Computern und ihren Unterschieden; die dann folgenden Programme in Maschinensprache laufen ohne Änderung meist auf dem KIM-1 (zum Teil mit dem Siebensegment-Display, zum Teil auch mit dem ASCII-Terminal oder Baudot-Fernschreiber).

In den meisten Fällen lassen sich die Programme leicht an andere Systeme, wie AIM-65 oder SYM-1, anpassen. Ein paar Stichworte aus dem Inhalt: Binär-Dezimal-Umwandlung, ASCII-Eingabe per Interrupt, Baudot-Ausgabe-Routine, Disassembler, Karteiprogramm, Textformatierung, Funktionsgenerator, Frequenzzähler, Debugger, Supermonitor... Es scheint, daß es fast nichts gibt, was man mit einem Mikroprozessor nicht

machen kann. Funkamateure finden in dem RPB-Band unter anderem ein komfortables Fernschreib-Programm, mit dem der KIM-1 das zu sendende Baudot-Signal direkt als Niederfrequenz ausgibt.

Wer sich erst etwas in die Materie einarbeiten will, findet u. a. eine übersichtliche Aufstellung der hexadezimalen 6502-Operationscodes, eine „Gebrauchsanleitung“ für programmierbare Timer, eine Beschreibung der CPU selbst und eine Aufstellung von Maschinenbefehlen, die – obwohl sie die Hersteller nicht nennen – zusätzlich noch möglich sind. Wer auch einmal einen Lötkolben in die Hand nimmt, kann dem Buch Schaltungen für die Hardware-Adressenverschiebung, für ein FSK-Modem, für einen A/D- und D/A-Wandler und einiges mehr entnehmen. Der Autor Herwig Feichtinger gab sich alle Mühe, ausgetüftelte Programme und nützliche Hardware in rund 120 Seiten zu pressen. (ISBN 3-7723-1731-6, DM 7.80)



BASIC ist eine Programmiersprache, die förmlich dazu verleitet, Programme „zusammenzuschustern“, ohne lange darüber nachzudenken, wie man dies und jenes effizienter implementieren könnte. Dabei genügt oft die Beachtung einiger weniger Grundregeln für das Programmieren, um Speicherraum und Laufzeiten einzusparen.

# Effiziente BASIC-Programmierung

## Arbeitsweise eines BASIC-Interpreters

Praktisch alle heute üblichen „Personal Computer“ arbeiten interpretativ, d. h. die BASIC-Programmierbefehle werden – Zeile für Zeile – als Folge von CPU-Maschinenbefehlen interpretiert und ausgeführt. (Im Gegensatz hierzu setzen „Compiler“ die BASIC-Befehle vor der Programmausführung in ein Maschinenprogramm um.)

Das „Tiny BASIC“ von Tom Pittman, das oft als Erweiterung für Einplatinen-Computer verwendet wird und etwa 2 KByte für den Interpreter einnimmt, speichert die eingegebenen Befehle direkt als ASCII-Zeichen im RAM, d. h. das Wort GOTO belegt vier Bytes. Komfortablere Interpreter, z. B. die BASIC-Versionen von Microsoft mit 8 KByte (PET, AIM usw.), setzen die Befehle zunächst in einen Zwischencode um, so daß jeder BASIC-Befehl nur noch ein Byte im Speicher belegt. Alle anderen Worte, z. B. Variablennamen, Kommentare oder Strings, werden allerdings auch hier im ASCII-Format gespeichert – ein Byte pro Zeichen. Bei der Programmausführung wird der Befehls-Zwischencode wiederum interpretativ abgearbeitet.

Die interpretative Arbeitsweise ist dafür verantwortlich, daß BASIC-Programme meist deutlich langsamer arbeiten als vergleichbare Maschinenprogramme. Dies kann jedoch bei vielen Rechen- und Textverarbeitungsaufgaben zugunsten einer leichteren Programmierbarkeit hingenommen werden. Lediglich bei zeitkritischen Aufgaben, z. B. bei der Steuerung von Peripheriegeräten oder bei solch komplexen Dingen wie einem Schachspiel, wird man auf die Möglichkeit zurückgreifen, Unterprogramme in der prozessor-spezifischen Maschinensprache zu verwenden und eventuell mit SYS oder USR vom BASIC-Programm her aufzurufen.

## Schnellere Programme

An einem kleinen Beispiel wollen wir uns einmal ansehen, wie man bei BASIC-Programmen die Geschwindigkeit erhöhen kann. Die dabei angegebenen Ausführungszeiten beziehen sich auf das 6502-Microsoft-BASIC des AIM-65, das etwa mit dem des PET 2001 übereinstimmt. Nehmen wir an, wir hätten folgendes Programm vorliegen:

```
10 FOR I=0 TO 20000
15 REM SCHLEIFE
20 NEXT I
```

Starten wir es mit RUN, so meldet der Computer nach etwa 36 Sekunden, daß er damit fertig ist. Eine erste Maßnahme zur Verringerung der Ausführungszeit (und des Speicherbedarfs) ist es, alle Leerräume wegzulassen:

```
10 FORI=0TO20000
15 REM SCHLEIFE
20 NEXTI
```

Ein Versuch zeigt, daß sich die Laufzeit nur etwa um eine Sekunde auf 35 s verkürzt hat – eine kaum nennenswerte Verbesserung. Der nächste Schritt bringt schon mehr: Wenn wir die Zeile 15 mit der REM-Anweisung weglassen, meldet sich der Computer schon 27 s nach RUN wieder. Eine weitere Verbesserung können wir erzielen, wenn wir schreiben:

```
10 FORI=0TO20000
20 NEXT
```

Jetzt benötigt das Programm nur noch 22 s – einfach durch Weglassen von I nach NEXT!

Bei „Tiny BASIC“ läßt sich das Programmbeispiel leider nicht nachvollziehen, da es hier keine FOR-NEXT-Anweisung gibt. Die (gleichwertige) Schleife

```
10 I=0
15 I=I+1
20 IF I < 20000GOTO15
30 END
```

dauert in Tiny BASIC auf einem 6502-System knapp zehn Minuten – eine Folge des direkten Interpretierens ohne Zwischencode.

Benötigt man z. B. die Zahl  $\pi$  mehrmals in einem Programm, so ist es zweckmäßig, diese Zahl irgendeiner Variablen zuzuweisen und später die Variable statt der Konstanten aufzurufen. Folgendes Programmbeispiel benötigt eine Ausführungszeit von etwa 17 Sekunden auf dem AIM-65:

```
10 FOR I=0 TO 500
20 B=3.14159265
30 NEXT
```

17 Sekunden werden also benötigt, die Konstante 3,14159265 fünfhundertmal der Variablen B zuzuweisen und sie dabei in das interne Fließkomma-Format umzurechnen. Die folgende, gleichwertige Programmversion läßt die Ausführungszeit auf etwa 1,2 Sekunden schrumpfen:

```
5 A=3.14159265
10 FOR I=0 TO 500
20 B=A
30 NEXT
```

Die Umrechnung in das Fließkomma-Format muß hier nur einmal geschehen, und 500mal wird der Variablen B der Wert der Variablen A (und damit wiederum  $\pi$ ) zugewiesen.

Dieser Programmiertrick ist allerdings nur dann sinnvoll, wenn es sich wirklich um eine Fließkomma-Konstante handelt. Will man dagegen einer Variablen solche Werte wie 0, 1, 2 oder 5 zuweisen, so ergibt sich kein Geschwindigkeitsvorteil der zweiten Programmversion.

Bei Tiny BASIC ist es in bezug auf die Ausführungsgeschwindigkeit günstiger, die Form LET A=... zu verwenden, da der Befehl dann schneller decodiert wird, als wenn man LET wegläßt. Bei anderen BASIC-Interpretern spielt dies keine Rolle, da mit und ohne LET ein gleichwertiger Zwischencode erzeugt wird.



Einige nützliche Kleinigkeiten: Der Ausdruck  $A+A$  wird schneller berechnet als  $2*A$ , weil die Multiplikation langsamer funktioniert als die Addition. Benötigt man innerhalb eines Programmes mehrmals z. B.  $SIN(1.5)$ , so sollte man diesen Wert nicht jedesmal neu errechnen, sondern einer Variablen zuweisen, die später beliebig oft aufgerufen werden kann. Das Schreiben mehrerer Befehle in eine statt getrennte Zeilen wirkt sich auf die Geschwindigkeit praktisch nicht aus.

Bei der Abfrage von I/O-Leitungen verwendet man statt des PEEK-Befehls mit nachfolgendem Vergleich besser den Befehl WAIT, wenn auf einen bestimmten Logikzustand gewartet werden soll. Seltsamerweise enthält das PET-Handbuch keinen Hinweis auf diesen Befehl, obwohl er auch beim PET funktioniert.

### Weniger Speicherplatz

Die Verwendung von LET ist bei Tiny BASIC zwar für die Geschwindigkeit vorteilhaft, kostet aber mehr Speicherplatz; hier muß der Programmierer selbst entscheiden, wo er Prioritäten setzt. REM-Kommentare im Programm kosten Zeit und Speicherplatz, erleichtern aber das spätere Verständnis für den logischen Ablauf. Ebenso schaffen Leerräume ein übersichtlicheres Programm, verschwenden aber je ein Byte. Ein „ausgekochtes“ Programm geht also immer zu Lasten der Verständlichkeit.

Die Verwendung der END-Anweisung ist bei Tiny BASIC stets nötig, bei

anderen Interpretern nur vor Unterprogrammen. Außerdem ist es möglich, in eine Zeile mehrere, von einem Doppelpunkt getrennte Befehle zu schreiben, so daß weniger Speicherplatz für die Zeilennummern benötigt wird. Selbstverständlich sollte man auch darauf achten, Speicher für ein- und mehrdimensionale Variablen am Anfang des Programms den wirklichen Erfordernissen nach zu reservieren und auch die Null-Indices, z. B.  $A(0)$ , zu verwenden, da auch sie erlaubt sind. Nach Möglichkeit sollte man Variablenamen mehrmals im Programm verwenden, wenn es nicht darauf ankommt, den alten Wert zu retten; auf diese Weise braucht kein zusätzlicher Speicherraum reserviert zu werden. Wenn zum Beispiel die Variable  $K\$$  verwendet wird, um das Tastenfeld mit dem GET- oder INPUT-Befehl abzufragen, so kann dies später im Programm wieder mit  $K\$$  geschehen.

Computer wie der PET 2001 oder der TRS-80 gestatten für viele Befehle Abkürzungen, z. B. ein Fragezeichen statt „PRINT“. Diese Abkürzungen haben auf den Speicherplatzbedarf der Programme keinerlei Auswirkung, da der gleiche Zwischencode wie bei der „normalen“ Befehlsform erzeugt wird; sie verkürzen lediglich die Eingabe. Listet man ein so eingetipptes Programm auf, erscheinen die Befehle wieder in voller Länge.

### Benutzerfreundlichkeit

Programme, die als Selbstzweck oder nur zur Verwendung des Programmierers selbst geschrieben wurden, können

bedenkenlos nach den oben gegebenen Ratschlägen optimiert werden. Programme, die auch in anderer Leute Hände gelangen können, erlauben diese Durchforstung aber nur in begrenztem Umfang, besonders, was die Verwendung von REM-Kommentaren angeht.

Ein wichtiger Aspekt der Benutzerfreundlichkeit ist die Möglichkeit von BASIC, im Dialog mit dem „Operator“ zu kommunizieren; dies sollte man auch ausnutzen. Wenn man Programme schreibt, bei denen bestimmte Funktionen nur mit Schlüsselworten ausgelöst werden können, so sollte man diese Schlüsselworte und ihre Bedeutung am Anfang des Programms erst einmal auf den Bildschirm schreiben, um die „Spielregeln“ zu definieren. Bei komplexen Programmen, die den Computer u. U. minutenlang beschäftigen, sollte man dem Benutzer mitteilen, daß es etwas dauern wird, um zu vermeiden, daß dieser voller Ungeduld die Break-Taste drückt und wieder von vorne anfangen muß. Allerdings: Je komfortabler dieser Dialog-Betrieb wird, desto mehr Speicherplatz belegt das Programm und desto langwieriger wird auch für den Benutzer der Umgang mit ihm. Möge jeder sein Maß finden!

Herwig Feichtinger

### Stichworte zum Inhalt

Speicherplatz, Geschwindigkeit, Dialogbetrieb, Variablenzuweisung.

## Mikrocomputer-Störstrahlung

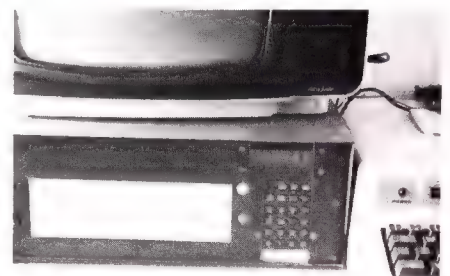
Stellt man z. B. ein kleines Mittelwellen-Radio neben einen Mikrocomputer, so stellt man schnell fest, daß auf allen möglichen Frequenzen (und nicht nur auf den Vielfachen der CPU-Taktfrequenz) starke Störungen auftreten. Die heute auf dem Markt befindlichen Mikrocomputer schneiden hier allerdings recht unterschiedlich ab: Der PET 2001 gehört zu den Geräten, die am wenigsten Störungen verursachen; dagegen können der TRS-80 und viele Einplatinen-Computer erhebliche Probleme hervorrufen, die besonders den Funkamateuren zu schaffen machen, die mit dem Mikrocomputer z. B. funkfern-schreiben oder ein Programm zum Decodieren von Morsezeichen ausprobieren wollen.

Bei Einplatinen-Computern wie dem KIM-1 (Bild) oder dem AIM-65 empfiehlt sich daher, ein Metallgehäuse zu verwenden; Siemens schlug mit seinem

PC-100 diesen Rat leider in den Wind. Auch eine Verdrosselung der Netzzuleitung ist nützlich und erfüllt noch einen zweiten Zweck, nämlich zu verhindern, daß aus dem Netz kommende Störimpulse in das Mikrocomputer-System gelangen und ein gerade laufendes Programm durcheinanderbringen; dies wurde z. B. beim AIM-65 schon mehrfach beobachtet. Netzfilter sind in der professionellen Technik inzwischen absolut üblich geworden und als fertige Module mit Löt- oder Schraubanschlüssen im Elektronik-Fachhandel erhältlich.

Etwas problematischer ist leider die Abblockung von I/O-Anschlüssen des Computers, da jede kapazitive Belastung zu einer unerwünschten Verringerung der Schaltgeschwindigkeit führt und auch induktive Verdrosselungsmaßnahmen wegen der dann auftretenden Induktions-Spannungsspit-

zen vermieden werden müssen. Die Zuleitungen zu den I/O-Ports sollen deshalb möglichst kurz sein, und eventuell notwendige Interface-Schaltungen (Leistungstransistoren, Optokoppler usw.) bringt man zweckmäßig noch im Mikrocomputer-Gehäuse unter.



Sinnvollerweise baut man Mikrocomputer – wie hier den KIM-1 – in ein Metallgehäuse ein, um ihre „elektromagnetische Verträglichkeit“ zu verbessern.



# Wanzenjagd

## Über die Fehlersuche in BASIC-Programmen

Zwar ist es richtig, daß man auf keine andere Weise so schnell mit einer Computersprache vertraut wird, wie durch die Fehlersuche in selbstgeschneiderten Programmen, zwar stimmt weiter, daß der Hobbyprogrammierer selbstentwickelte Verfahrensweisen beim Entwanzen in aller Regel sicherer zu handhaben weiß als solche, die ihm in der Literatur nahegelegt werden – aber der lange und beschwerliche Weg bis zur Beherrschung einer betriebssicheren Fehlersuche zwingt so manchen Anfänger oft genug, das Rad gewissermaßen noch einmal neu zu erfinden, und ein derart mühsames Geschäft ist nicht nach jedermanns Geschmack. Und so erlaubt sich der Autor denn hier, den Versuch zu machen, einige jener Erfahrungen mitzuteilen, die er beim Entwickeln seiner Fehlersuch-Praktiken erwarb.

Ausgangslage: Es schreibt ein Nicht-Profi mit absolut nichttechnischem Beruf, der im Frühjahr 1978 seine ersten Gehversuche mit einem neuerstandenen PET unternahm, und zu diesem Zeitpunkt lediglich über einige Erfahrungen mit programmierbaren Taschenrechnern sowie dem berühmten Microset 8080 verfügte, im übrigen aber von Computern nicht sehr viel mehr wußte, als man üblicherweise aus der Populärliteratur und Elektronik-Fachzeitschriften erfährt. Diese persönliche Einschlebung sei erlaubt, weil sie die Frage der Legitimation dessen berührt, der hier gute Ratschläge geben will – es geht hier nicht um Hinweise durch einen Super-Spezialisten mit jahrelanger Berufserfahrung. Also: Tips eines Steckenpferd-Reiters für Mit-Hobbyisten, mit denen keinesfalls der Anspruch verbunden ist, den Stand der Wissenschaft wiederzuspiegeln – nun zur Sache:

Fehlersuche beginnt, so merkwürdig das klingen mag, bereits vor dem Einschalten des Rechners, bei der ersten Grobkonzeption des Programms näm-

lich. Wer von Anfang an darauf achtet, daß er sein Programm in wohldefinierte Segmente von nicht mehr als höchstens zwei Dutzend Zeilen aufteilt und sauber definiert, was in diesen „Programm-Moduln“ zu geschehen habe, für den ist ein beträchtlicher Teil der ohne diese strukturierte Programmierung ansonsten fälligen Fehlersuche bereits gelaufen.

Wohldefinierte Segmente – was heißt dies? Neben der eigentlichen Aufgabendefinition gehört zu den Notizen, die sich der Programmierer zu jedem einzelnen Modul zu machen hat, eine Beschreibung der „Datenquellen“ für das Segment; Variable stehen hier an erster Stelle, aber auch Eingaben von der Tastatur oder das Einholen der Informationen von Band, Floppy oder einem Lochstreifenleser gehörten hierher. Danach legt man die „Daten-Senke“ fest, d. h. den Ort, an dem das Programmsegment das Ergebnis seiner Bemühungen zu deponieren hat; wiederum eine oder mehrere Variable in den meisten Fällen, häufig aber auch der Bildschirm, Drucker, die Floppy, das Tonband, und als Sonderfall der interne Programmzähler, denn es gibt Programmsegmente, deren Funktion nicht unmittelbar die Produktion neuer Daten ist, sondern die als „Weiche“ (Program Switch) dienen und in Abhängigkeit vom Ergebnis einer im Programmmodul enthaltenen Prüfung zu anderen Segmenten verzweigen.

Befinden sich Prüfung und Weiche nicht im gleichen Segment, wird man eine Variable als „Zwischennotiz“ benutzen – etwa in einem Spiel den Namen „R“ für Reihenfolge, und dann festlegen: Wenn R gleich 0, dann ist der Rechner an der Reihe, zu spielen, wenn R gleich 1, dann der Spieler. Man nennt eine solche Variable, in der nicht eigentlich Zahlen gespeichert werden, sondern (durch Zahlen dargestellte) Sachverhalte, ein „Flag“, was soviel wie Markierungswimpel heißt und

wohl aus dem Pfadfinder-Leben entlehnt wurde. Flags also können ebenfalls Datenquelle oder Datensenke sein.

Fassen wir an dieser Stelle einmal zusammen: Bereits bei der Grobkonzeption haben wir unser Programm in handliche und – hierauf kommt es an – bequem zu prüfende Module zerlegt, haben festgelegt, was die Aufgabe des Segments ist, welches „Material“ es zur Erledigung der Aufgabe bekommt, wie das Produkt auszusehen hat und wo es abzuliefern sei. Zur ausreichenden Charakterisierung des Programmmoduls gehört noch eine Liste jener Variablenamen, die eigentlich nicht zur Datenübergabe an das Modul bestimmt sind, deren Inhalt aber Einfluß auf ein ordentliches Arbeiten eben diese Segments hat (Parameter-Variable).

Dies zur Buchführung über die Programmsegmente; nun zu zwei Listen, die jetzt anzulegen sind: ein „Verzeichnis der Variablen“ und ein „Katalog der Unterprogramme“. Im Variablen-Verzeichnis notieren wir den Variablen-Namen, die Funktion (beispielsweise „Trefferzahl von Spieler 1“) und den Ort der Verwendung. Diese Liste ist später nicht nur ein höchst nützliches Werkzeug zur Fehlersuche, sondern sie hilft auch, Speicherplatz zu sparen: Wenn Variable nur lokal benötigt werden, können wir sie mehrfach belegen. Das letzte Papier, das beim Programm-entwurf auf dem Schreibtisch liegen muß ist der Unterprogramm-Katalog. Hier notieren wir außer den Angaben, die ohnehin zu jedem Programmsegment gehören, zusätzlich noch, von welchen Zeilen das Unterprogramm aufgerufen wird und auf welcher Ebene. Unterprogramme der Ebene eins werden vom Hauptprogramm gerufen, solche der Ebene zwei ihrerseits von Unterprogrammen der Ebene eins – und so fort.

Soviel Bürokratismus – wozu Dinge aufschreiben, die man sich ohnehin merken kann, wird der Leser an dieser



Stelle möglicherweise fragen. Nun, glauben Sie dem Autor: All diese zeitraubenden Aufzeichnungen dienen dazu, Zeit zu sparen. Sie verhindern, daß man bereits im Frühstadium einer Programmstehung Fehler einbaut, die später nur mühsam zu entfernen sind und stundenlanges Suchen erfordern. Die Erfahrung zeigt, daß man gerade bei den „selbstverständlichen“ und „offensichtlichen“ Sachverhalten Irrtümer begeht.

Nun, einmal unterstellt, der Programmierer habe sich bis jetzt an die obigen Ratschläge gehalten und brav Listen und Verzeichnisse geführt – dann kann nun von der in freier Form notierten Modulbeschreibung zur Formulierung in Basic übergegangen werden. Programme bis zur Größe von 3 bis 4 KByte wird man ohne Einzeltest der Module nun unmittelbar programmieren, für besonders schwierige Segmente oder für solche, denen der Programmierer von sich aus nicht recht traut, empfiehlt sich auf jeden Fall eine Serie von Probelaufen. Zumindest ein übersichtlicher Test jedes komplexeren Programm-Moduls ist an dieser Stelle auch dann ratsam, wenn das begonnene Projekt einen Umfang von 4 KByte übersteigt.

Segmenttest – aber wie? Jetzt kommt uns das Ergebnis der vorangegangenen Papierschlacht erstmals zugute. Anhand der Aufzeichnungen setzen wir von Hand die „Datenquellen“ auf ihre Ausgangswerte und fahren unser Programmsegment – das zuvor durch eine vorübergehend eingebrachte „Return-Zeile“ abgeschlossen wurde – mit dem „Gosub“-Befehl ab, dem die Zeilennummer der jeweils ersten Segment-Zeile folgt. Nach diesem Testlauf fragen wir die Datensinken von Hand aus ab und prüfen, ob das Verarbeitungsergebnis den Erwartungen entspricht. Wenn dies nicht zutrifft, verkleinern wir durch Verlegen des Return-Befehls in Richtung niedrigerer Zeilennummern oder durch Verlegen des Startpunktes in Richtung höherer Zeilennummern unseren Prüfbereich, bis wir den Ort lokalisiert haben, an dem das Malheur passiert; die Reparatur fällt nun nicht mehr schwer. Je nach Komplexität des Segments wird es erforderlich sein, den Test mit geänderten Verhältnissen in der Datenquelle noch einige Male zu wiederholen und hierbei auch die Parameter-Variablen zu variieren.

Sind alle Segmente durchgeprüft (und die nun überflüssigen Return-Anweisungen entfernt), kann man das Programm nun erstmals in voller Länge laufen lassen. Liegen keine Denkfehler des Programmierers vor, die den generellen Ablauf betreffen und erst durch das Zusammenwirken der Segmente sichtbar werden, dann wird – aufgrund

## Variablenliste

Name	Funktion	Bedeutung	Benutzt in:
G	Winkel-Flag	G = 1: Winkel in Grad G = 0: Winkel in Radiant	200–400
E	Eingabe-Flag	E = 0: Eingabe gesperrt	1100–1200
A	Zwischenakku	Kurzzeit-Werte	100–200 600–680 2000–3000
A %	Eingabe-Zähler	Zeichenzahl im String	300–380
F1	Anfangsfrequenz		300–400
F2	Arbeitsfrequenz		300–400
F3	Endfrequenz		350–500
F3	Altfrequenz	Rechnungswiederholung	510–990
C	Kapazität	Schwingkreis-C	420–480
L	Induktivität	Schwingkreis-L	420–480
		... (und so fort)	

So kann die „Variablenliste“ gestaltet sein, die der Autor des Beitrags „Wanzenjagd“ zur Programmanalyse und Fehlersuche empfiehlt

## Beispiel einer „Programmsegment-Beschreibung“

**Aufgabe:** Errechnung der Resonanzfrequenz eines Doppel-T-Filters als Funktion der Kondensatorkapazität

**Datenquelle:** C

**Datensenke:** F

**Parameter-Variable:** R

**BASIC-Text**  $F = 1/(2 \cdot \pi \cdot R \cdot C)$

der gründlichen Vorarbeit – das Programm zufriedenstellendes Verhalten zeigen.

Tut es das wider Erwarten nicht oder hat man einen Fehler in einem Programm zu suchen, das nicht von vornherein dem Gesichtspunkt der Service-Freundlichkeit entsprechend entworfen wurde, dann empfiehlt sich ein anderes Verfahren: Man geht vom Fluß-Diagramm aus (wo dieses nicht vorliegt, ist es nach einer Programm-Analyse anzufertigen) und führt nun nachträglich auf dem Papier eine Einteilung in Segmente durch, wobei man zuerst einmal von Großblöcken ausgeht, die man später verkleinert. An das

Segment-Ende fügen wir nun im Programm den Befehl „STOP“ ein. Jedesmal, wenn der Rechner an diesen Unterbrechungspunkt kommt, meldet sich das Betriebssystem, und die Abarbeitung des Benutzer-Programms ruht.

Wir können nun per Hand die betreffende Datensenke abfragen und das Programm dann durch die Befehle „CONT“ beziehungsweise „CON“ – sie hängen vom verwendeten Basic-Dialekt ab – fortsetzen. Auch hier gilt: Durch Verlagerung des Unterbrechungspunktes wird der Fehler mehr und mehr eingekreist. Auch in diesem Fall ist genaue Buchführung über jede Aktion und jede Reparatur erforderlich, denn es könnte ja sein, daß eine lokale Korrektur zwar zum befriedigenden Arbeiten des Segmentes führt, dafür aber andernorts ein neuer Fehler auftritt, weil die Art der Reparatur nicht kompatibel mit den Anforderungen später durchlaufener Programmsegmente war – das Konzept der Aufzeichnung von Datensenke und Datenquelle gibt das Werkzeug an die Hand, auch solchen heimtückischen Serienfehlern auf die Spur zu kommen. Halali, die Jagd geht auf: Viel Glück bei der Wanzensuche!

## Wo steht etwas über...

Am Schluß der meisten Beiträge dieses Sonderheftes sind „Stichworte zum Inhalt“ zu finden. Wenn bei Ihnen schon das Zeitalter der elektronischen Datenverarbeitung begonnen hat, und Sie Ihren Computer dazu benutzen, nach Literaturstellen zu suchen (ein entsprechendes Programm erschien unter dem Titel „KIM auf Datensuche“ in FUNKSCHAU 1978, Heft 24), dann können Sie Ihre elektronische Literatur-Kartei sehr ein-

fach durch Eintippen des Beitrags-Titels und der Stichworte auf dem Laufenden halten. Der Computer kann dann in Sekundenschnelle zu beliebigen Stichworten wie etwa „ASCII“, „6502“, „Parity“ oder „Mikroprogramm“ den passenden Beitrag ausgeben. Beim AIM-65 ist das besonders einfach: Er hat ein geeignetes Suchprogramm schon in seinem ROM-Editor eingebaut; es läßt sich mit „F“ aufrufen.



Hans-Georg Joepgen

Daß Maschinenprogramme wesentlich schneller sind als solche in interpretierten Hochsprachen wie Basic, Pascal und Focal, gehört inzwischen fast zum Grundwissen des Programmierers. In welcher Größenordnung jedoch der Gewinn an Rechengeschwindigkeit liegt, wenn man auf Maschinensprache umsteigt, darüber herrscht vielerorts noch Unklarheit. Das im folgenden vorgestellte Beispiel betrifft ein und denselben Job, der jedoch in drei verschiedenen Sprachen programmiert wurde, wobei Rechenzeit-Unterschiede bis zum rund Vierhundertfachen zu registrieren waren.

# Suchprogramm

– formuliert in drei Sprachen

Bei der Bearbeitung eines ursprünglich in Kassettenversion vorliegenden FOCAL-Interpreters, der in das DOS des Euro-Apple integriert werden sollte, ergab sich die Notwendigkeit, wiederholt jene Adresse festzuhalten, an der in einem knapp 6000 Byte tiefen Speicherbereich eine bestimmte, fünf Byte lange Zeichenkette abgespeichert war. Hierzu diente zuerst einmal ein in Applesoft geschriebenes Programm – eine Abart jenes Basic-Dialektes von der amerikanischen Firma Microsoft, auf die auch das Level-II-Basic des Rechners TRS-80 und die Sprache Commodore-Basic des PET zurückgehen. Bild 1 zeigt das Listing dieses Programms: Nach einem Befehl in Zeile 14, der eine Anweisung zur Speicherverwaltung enthält, beginnt eine Schleife mit dem Zähler N, in der jeweils die fünf Speicherstellen mit den Adressen N bis N plus 4 ausgelesen werden. Der Inhalt wird in einer Zeichenkette A-String abgelegt und anschließend mit der Musterkette verglichen, deren Adresse man sucht – hier zur Demonstration die zum Wort FOCAL gehörenden ASCII-Zeichen. Bei Gleichheit beider Ketten meldet der Rechner dies unter Angabe der Adresse, bei Ungleichheit wird die Arbeit fortgesetzt, bis der zu untersuchende Adreßbereich vollständig durchgeprüft ist. Für dieses Programm ergab sich eine Laufzeit von 384 Sekunden – viel zu viel für den gewünschten Zweck.

Zur Verkürzung der Rechenzeit wurde zunächst einmal versuchsweise die Ausgaberroutine für Blindadressen, die anfänglich zu Kontrollzwecken eingebaut worden war, entfernt – allein damit ging die Rechenzeit auf 144 Sekunden und damit auf rund ein Drittel zurück (Bild 2).

Dies konnte jedoch noch immer nicht befriedigen, worauf folgende Änderungen vorgenommen wurden: Wechsel

von Fließkomma-Basic in einen Ganzzahl-Basic-Dialekt „Apple Basic“, Verzicht auf die Behandlung der Speicherinhalte als Glieder einer Kette und stattdessen Einzelprüfung durch Vergleich des unmittelbaren Byte-Wertes sowie, als dritte Maßnahme, Einbau eines sofortigen Absprungs bei Nicht-Koinzidenz auch nur eines einzigen Zeichens. Der Gewinn an Verarbeitungsgeschwindigkeit war frappierend; jetzt lief der Job nur noch 27 Sekunden, ge-

Bild 1. Programmversion 1, geschrieben für den Applesoft-Interpreter. Das Programm läuft 384 Sekunden

```
10 REM FCL-UNTERSUCHUNG 0
12 REM -----
14 HIMEM: 28670: REM $6FFE
20 FOR N = 28672 TO 34405
25 PRINT N
30 A$ = CHR$(PEEK(N)) + CHR$(
    PEEK(N + 1)) + CHR$(PEEK
    (N + 2)) + CHR$(PEEK(N +
    3)) + CHR$(PEEK(N + 4))
40 IF A$ = "FOCAL" THEN PRINT "
    GEFUNDEN AB DEZADRESSE "N
50 NEXT : PRINT " +++ BEENDET. "
```

genüber der ursprünglichen Laufzeit mithin ein Gewinn um mehr als das Vierzehnfache. Zum Programm-Listing (Bild 3) ist am Rande zu bemerken, daß der zu untersuchende Focal-Interpreter hier in einen anderen Speicherbereich gelegt werden mußte, weil Apple Basic den Adreßraum Dezimal 28 670 bis 34 405 teilweise selbst belegt. Das Doppelkreuz in den Zeilen 30 bis 70 ist die Apple-Basic-Entsprechung zum Zeichen <> der Microsoft-Basic-Dialekte und bedeutet „ungleich“.

Der entscheidende Durchbruch zu annehmbaren Verarbeitungszeiten gelang jedoch erst durch Ausweichen auf

Maschinensprache. Mit dem in Bild 4 aufgelisteten Programm ergibt sich für die Bearbeitung der gleichen Aufgabe eine Rechenzeit von etwas weniger als eine Sekunde!

Vor dem Start werden die ersten sieben Speicherplätze der Seite 0 mit folgenden Werten geladen: Länge der Musterkette, L-Byte der Adresse, an der die Musterkette abgelegt ist, H-Byte dieser Adresse, und – jeweils zuerst das L-Byte, gefolgt vom höherwertigen Byte –

die Anfangs- und die Endadresse des Speicherfeldes, das durchsucht werden soll. Sodann wird die Musterkette selbst geladen: hier ab Adresse 4059 (alle Adressen im Hexcode).

Nach dem Start ab Adresse 4000 lädt der Rechner die Kettenlänge aus Speicherplatz 0000 in das Indexregister Y und greift dann mit der Adressierungsart indirekt-indiziert auf jene Speicherzelle zu, deren Adresse der Rechner enthält, indem er zu dem in den Speicherzellen 3 und 4 gespeicherten Doppel-Byte den Inhalt des Y-Registers addiert. Der mit der so gewonnenen Adresse angesprochene Wert wird in



den Akku geladen und dann mit der Musterkette verglichen, auf die der Rechner ebenfalls indirekt-indiziert zurückgreift. Bei Gleichheit erfolgt Absprung nach Zeile 4035. Dort vermindert der Rechner den Inhalt des Y-Registers um 1 und prüft es auf Null. Ist der Inhalt von Y noch ungleich Null, so ist vorerst nur ein Teil der Musterkette mit dem gerade untersuchten Speicherbereich inhaltsgleich; es erfolgt ein Rücksprung zur weiteren Untersuchung nach 4002. Lag als Y-Inhalt dagegen Null vor, so war die Musterkette in voller Länge deckungsgleich mit dem Speicherfeld. Der Rechner schreibt die Zeichen „OK“ auf den Schirm, gefolgt von einem Doppelpunkt und der Adresse, an der die Kette gefunden wurde. Hierbei leisten zwei Unterprogramme aus dem Betriebssystem des

Bild 3. ►  
Benutzung eines Ganzzahl-Interpreters und Änderung der Programmstruktur bringt einen neu-erlichen Gewinn an Verarbeitungsgeschwindigkeit: Laufzeit nur noch 27 Sekunden

```
10 REM FCL-UNTERSUCHUNG 1
12 REM -----
14 HIMEM: 28670: REM $6FFE
20 FOR N = 28672 TO 34405
30 A$ = CHR$(PEEK(N)) + CHR$(
    (PEEK(N+1)) + CHR$(PEEK(N+
    2)) + CHR$(PEEK(N+
    3)) + CHR$(PEEK(N+4))
40 IF A$ = "FOCAL" THEN PRINT "
    GEFUNDEN AB DEZADRESSE "N
50 NEXT : PRINT " +++ BEENDET. "
JRUN
    GEFUNDEN AB DEZADRESSE 34379
+++ BEENDET.
```

Bild 2. Nach Entfernung der Kontroll-Ausgabe durchlaufener Adressen geht die Rechenzeit auf 144 Sekunden zurück – im Anschluß an das Listing die Erfolgsmeldung des Rechners

Euro-Apple Unterstützung: FDED druckt den Akkuinhalt in Form eines Zeichens, das nach einem internen und nichtgenormten Betriebscode entschlüsselt wird (Hex CF CB BA für OK). Die Subroutine mit Startadresse FDBD dagegen druckt den Akku-Inhalt als HEX-Doppelzeichen aus (also zum Beispiel 34 für das Bitmuster 0011 0100).

Zurück zu Zeile 4005; bei Ungleichheit erfolgt der oben beschriebene Absprung nicht. In diesem Fall erhöht der Rechner den Inhalt des Doppelbytes in den Speicherzellen 0003 und 0004, in die anfänglich die Startadresse des zu untersuchenden Feldes deponiert wurde, um 1. Das Doppelbyte dient als Pointer und weist, gemeinsam mit Y, jeweils auf die zu untersuchende Speicherstelle. Die Addition von 1 hat also die Funktion eines Fortschaltens dieses Zeigers. Nach der Inkrementierung wird geprüft, ob die Schlußadresse bereits erreicht ist, wenn nicht, erfolgt Rücksprung nach Programmspeicher-

zelle 4000 – wenn ja, druckt der Rechner „ENDE“ und ruft mit dem Befehl RTS wieder seinen Monitor auf.

Ein Beispiel also, das nicht nur zeigt, wie mit Hilfe von Maschinensprache-Programmen erhebliche Gewinne an Rechengeschwindigkeit zu erzielen sind, sondern auch, daß Programmieren in Assembler wesentlich mühsamer ist als das Programmdichten in Hochsprachen. Dennoch gibt es genügend Fälle, wo man den Komfortverzicht gern in Kauf nimmt, wenn „die Zeit drängt“: Siehe die Suche nach Fünfergruppen im Focal-Interpreter. –

(Vgl. auch den Beitrag „Rechengeschwindigkeit auf dem Prüfstand“.)

#### Stichworte zum Inhalt

FOCAL, BASIC, 6502, Apple-II, Suchprogramm, Maschinenprogramm, Rechengeschwindigkeit

```
10 REM FCL-UNTERSUCHUNG 2
12 REM -----
20 FOR N=3900 TO 9824
30 IF PEEK (N)#70 THEN GOTO 100
40 IF PEEK (N+1)#79 THEN GOTO
    100
50 IF PEEK (N+2)#67 THEN GOTO
    100
60 IF PEEK (N+3)#65 THEN GOTO
    100
70 IF PEEK (N+4)#76 THEN GOTO
    100
80 PRINT " GEFUNDEN AB DEZADRESSE "
    :N
100 NEXT N: PRINT " +++ BEENDET. "
    :END
JRUN
    GEFUNDEN AB DEZADRESSE 9803
+++ BEENDET.
```

Bild 4. ►  
Ein Programm zur Lösung der gleichen Aufgabe, diesmal geschrieben in Maschinensprache und aufgelistet vom Disassembler des Euro-Apple: Jetzt erledigt der Rechner den gleichen Job, für den er in der Programmfassung nach Bild 1 noch über 6 Minuten brauchte, in weniger als einer Sekunde

```
*4000LLL
4000- A4 00 LDY $00
4002- D8 CLD
4003- B1 03 LDA ($03),Y
4005- D1 01 CMP ($01),Y
4007- F0 2C BEQ $40035
4009- 18 CLC
400A- A5 03 LDA $03
400C- 69 01 ADC #$01
400E- 85 03 STA $03
4010- A5 04 LDA $04
4012- 69 00 ADC #$00
4014- 85 04 STA $04
4016- C5 06 CMP $06
4018- 30 E6 BMI $4000
401A- A5 03 LDA $03
401C- C5 05 CMP $05
401E- 30 E0 BMI $4000
4020- A9 C5 LDA #$C5
4022- 20 ED FD JSR $FDED
4025- A9 CE LDA #$CE
4027- 20 ED FD JSR $FDED
402A- A9 C4 LDA #$C4
402C- 20 ED FD JSR $FDED
402F- A9 C5 LDA #$C5
4031- 20 ED FD JSR $FDED
4034- 60 RTS
4035- 88 DEY
4036- D0 CA BNE $4002
4038- A9 CF LDA #$CF
403A- 20 ED FD JSR $FDED
403D- A9 CB LDA #$CB
403F- 20 ED FD JSR $FDED
4042- A9 BA LDA #$BA
4044- 20 ED FD JSR $FDED
4047- A5 04 LDA $04
4049- 20 BD FD JSR $FDBD
404C- A5 03 LDA $03
404E- 20 BD FD JSR $FDBD
4051- A9 A0 LDA #$A0
4053- 20 ED FD JSR $FDED
4056- 4C 02 40 JMP $4002
4059- 46 4F LSR $4F
405B- 43 ???
405C- 41 4C EOR ($4C,X)
405E- EA NOP
405F- EA NOP
*4000G
OK:264B ENDE
```



In vielen Ländern ist der Anschluß zumindest elektroakustisch angekoppelter Modems an einen Fernsprechapparat ohne Formalitäten gestattet; in der Bundesrepublik brauchen solche „Koppler“ eine FTZ-Nummer. Der folgende Beitrag geht auf übliche Techniken zur Datenübertragung ein.

# Datenübertragung per Telefon

## Bitrate

Die erzielbare Übertragungsgeschwindigkeit, gemessen in bit/s, wird in erster Linie durch die zur Verfügung stehende Bandbreite und Phasenverzerrungen innerhalb dieser begrenzt. Üblicherweise reicht der Frequenzumfang einer Fernsprech- oder Funkverbindung von 300...3000 Hz. Leider läßt er sich aber nicht ganz ausnutzen, da die Frequenz 2100 Hz zum Auslösen der sog. Echosperrre in den Vermittlungsstellen dient, die bei Fernverbindungen wirksam wird. Ferner ist bei akustisch angekoppelten Modems damit zu rechnen, daß enorme Phasen- und Amplitudenschwankungen über diesen großen Bereich hinweg auftreten. Aus diesem Grund hat sich in der Praxis gezeigt, daß akustisch gekoppelte Modems nur bis etwa 300 bit/s zuverlässig funktionieren, hier ist der Frequenzbandbedarf noch relativ gering.

## Duplex und Simplex

Je nachdem, ob die Verbindung in beide Richtungen gleichzeitig möglich sein soll oder nicht, spricht man von Duplex- oder Simplex-Betrieb; man benötigt beim Duplex-Betrieb also einen Rückkanal. „Halbduplex“ ist die wechselseitige Übertragung in beide Richtungen über einen Kanal.

Die bei Modems überwiegend verwendete Modulationsart ist FSK (Frequency Shift Keying), da sie recht störicher funktioniert. Jedem der beiden logischen Zustände 0 und 1 wird dabei eine Tonfrequenz zugeordnet, so daß sich pro Datenkanal ein Frequenzpaar ergibt. Bei Duplexverbindungen ver-



Bild 1. Frequenzspektrum nach der CCITT-Empfehlung V-21 für Duplexverbindungen bis zu 300 bit/s

wendet man nun zwei Frequenzpaare, die genügend weit auseinanderliegen und sich somit durch selektive Filter leicht voneinander trennen lassen.

Bild 1 zeigt das von der CCITT genormte Frequenzspektrum für Duplex-Modems bis zu 300 bit/s (V-21). Dabei ist genau festgelegt, wer welches Frequenzpaar zum Senden verwendet: Der anrufende Teilnehmer sendet mit dem Paar 980/1180 Hz, der gerufene sendet 1650/1850 Hz. Bild 1 macht auch deutlich, daß nicht nur die diskreten Frequenzen selbst belegt sind, sondern – wie bei jeder Modulation – auch ein gewisses Spektrum um sie herum. Die Frequenzen sind zusammen mit der Bitrate so gewählt, daß innerhalb eines Kanals eine relativ gleichmäßige spektrale Leistungsverteilung auftritt.

Für akustische Koppler hat sich bei Simplex-Übertragungen das Verfahren gut bewährt, das Frequenzpaar 1300/1700 Hz bei 300 bit/s zu verwenden. Selbstverständlich sind auch andere Frequenzen denkbar, aber nur die CCITT-Frequenzen bieten die Gewähr dafür, daß in den Fernsprech-Vermittlungsämtern nichts „passiert“.

## Praktische Ausführung von Modems

### Der Modulator

Der Modulator hat die Aufgabe, ein Digitalsignal in ein FSK-Nf-Signal umzuformen. Dazu dient meist ein Oszillator mit elektronisch umschaltbarer Frequenz. Bis zu 300 bit/s ist es nicht erforderlich, eine bestimmte Phasenbezie-

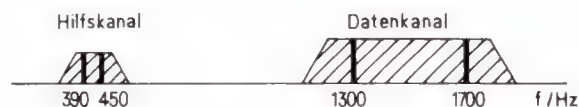


Bild 2. Frequenzspektrum nach CCITT V-23 für Simplex- oder Halbduplex-Verbindungen bis zu 600 bit/s

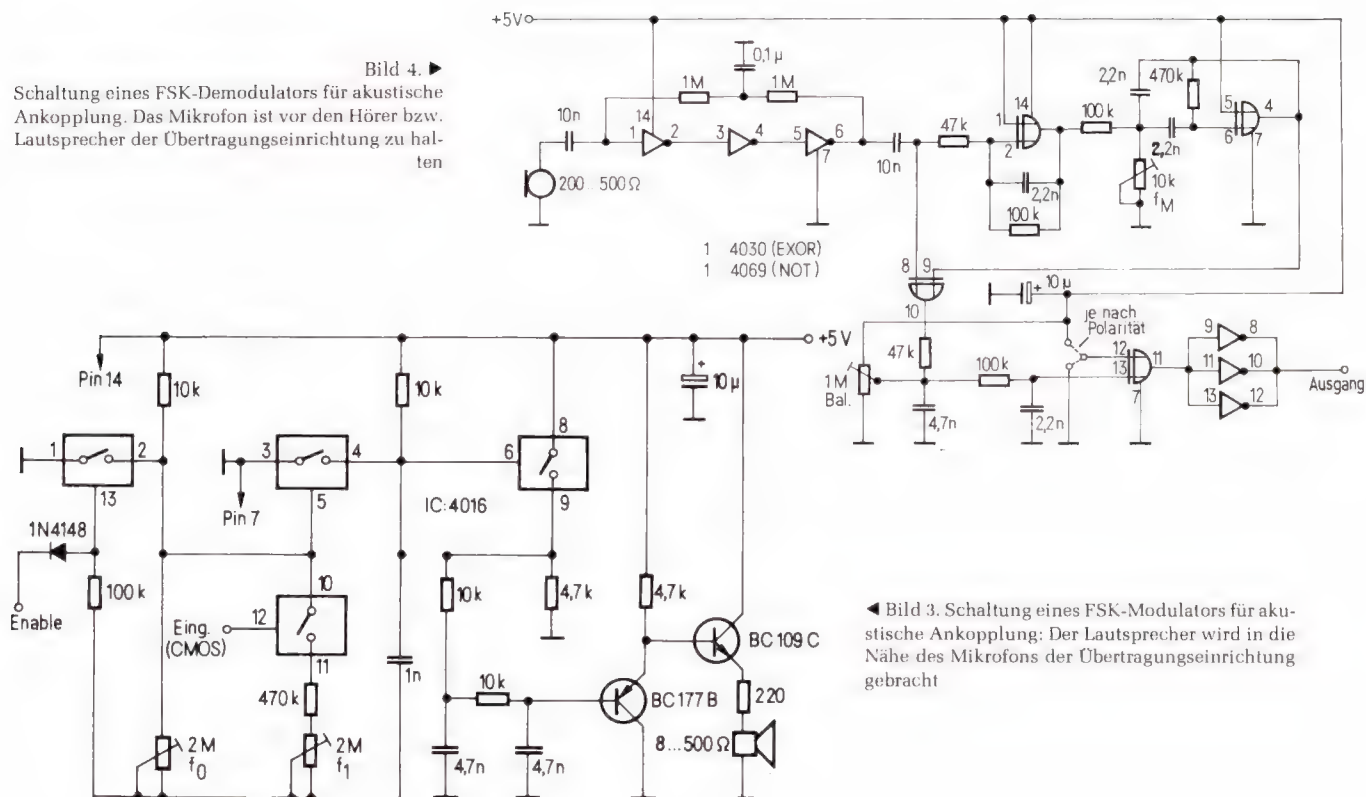
Beschränkt man sich auf Simplex-Betrieb, z. B. für die einseitige Übertragung von Programmen und Daten von Mikrocomputern, so kann man bei der Frequenzwahl von anderen Überlegungen ausgehen. Die Störsicherheit verbessert sich nämlich, wenn man die beiden FSK-Frequenzen weiter auseinanderlegt, und dies ist bei einkanaliger Übertragung ja ohne weiteres möglich. Die CCITT-Empfehlung V-23 sieht für Modems bis zu 600 bit/s daher die Frequenzen nach Bild 2 vor. (Für Geschwindigkeiten zwischen 600 bit/s und 1200 bit/s ersetzt man den 1700-Hz-Ton durch 2100 Hz, der dann allerdings mit dem Echosperrre-Antwortton zusammenfällt.) Ein schmaler Hilfskanal ( $\leq 75$  bit/s) dient zum Übertragen von Steuer- und Quittungssignalen, wird oft aber nicht verwendet.

hung zwischen Datentakt und Nf herzustellen, so daß Schaltungen nach Bild 3 völlig genügen. Der Oszillator selbst liefert ein Rechtecksignal, das durch einen Tiefpaß von seinen ungeradzahlgigen Harmonischen befreit wird. Der Ausgang kann einen kleinen Lautsprecher treiben, der die akustische Senderankopplung besorgt.

Es sei darauf hingewiesen, daß die CCITT-Empfehlungen zwar die Tonfrequenzen, nicht aber das zu verwendende Datenformat (ASCII, Baudot, Parity-Bits usw.) normen. Diesem Gebiet ist ein anderer Beitrag in diesem Heft gewidmet. Allgemein betrachtet man die höhere der jeweils verwendeten FSK-Frequenzen als log. 1, die niedrigere als Null. Beim Funkfern schreiben werden die Frequenzen für 1 und 0 als Mark und Space bezeichnet.



Bild 4. ►  
Schaltung eines FSK-Demodulators für akustische Ankopplung. Das Mikrofon ist vor den Hörer bzw. Lautsprecher der Übertragungseinrichtung zu halten



◄ Bild 3. Schaltung eines FSK-Modulators für akustische Ankopplung: Der Lautsprecher wird in die Nähe des Mikrofons der Übertragungseinrichtung gebracht

## Der Demodulator

Wenn die beiden FSK-Frequenzen nicht allzu weit auseinanderliegen, ist die Demodulator-Schaltung aus dem HOBBYCOMPUTER-Sonderheft 1 des Franzis-Verlages nicht mehr allzu gut geeignet. Besser bewährt hat sich eine Schaltung nach Bild 4, die für das Frequenzpaar 1300/1700 Hz ausgelegt ist und deren Eingangsempfindlichkeit für ein kleines dynamisches Mikrofon ausreicht. Damit ist eine akustische Ankopplung an beliebige Übertragungswege möglich. Am Ausgang steht das demodulierte Digitalsignal (bis zu 300 bit/s) mit CMOS-Pegel zur Verfügung.

Beide Schaltungen lassen sich zusammen recht gut testen, indem man den Lautsprecher des Modulators vor das Mikrofon des Demodulators hält oder auch die akustische Ankopplung an ein Tonbandgerät ausprobiert. Zum Abgleich der Modulator-Frequenzen sollten ein Frequenzzähler und möglichst ein Oszilloskop vorhanden sein.

Die beiden Trimpoties im Demodulator stehen normalerweise etwa in Mittelstellung; eines dient dem Abgleich auf die Mittenfrequenz (hier 1500 Hz), um den Phasenschieber richtig arbeiten zu lassen, das andere gestattet die Kompensation von Unsymmetrien der CMOS-Gatter.

## CMOS-ICs in Analoganwendungen

Wie sich in einigen Versuchen herausstellte, führt die in Bild 4 angegebene Schaltung ab und zu zu Schwierigkeiten, weil die Exklusiv-Oder-Gatter leichter zum Schwingen neigen als normale Inverter. Es ist daher empfehlenswert, die beiden oben rechts befindlichen EXOR-Gatter – sie sind hier invertierend geschaltet – mit den beiden davor befindlichen 4069-Invertiern zu vertauschen.

Ferner zeigt die Erfahrung, daß die CMOS-ICs der HEF-Reihe (LOC MOS) zwar ein günstiges Schaltverhalten besitzen, für Analoganwendungen aber kaum geeignet sind, weil ihre hohe Durchgangsverstärkung mangels interner Frequenzkompensation oft zum Schwingen auf einer hohen Frequenz führt. Dagegen funktionieren die CMOS-ICs anderer Hersteller auch bei 100%iger Gegenkopplung noch einwandfrei.

Der Stromverbrauch von linear betriebenen CMOS-Gattern hängt sehr stark von der Betriebsspannung ab; bei 3 V ist er nahezu Null und erreicht bei Spannungen von 10 V und mehr einige mA. Außerdem sinkt die Verstärkung mit zunehmender Betriebsspannung – ein Grund mehr, CMOS-ICs in Analogschaltungen mit nur etwa 5 V zu versorgen. In Bild 4 ergibt sich daraus außer-

dem noch der Vorteil, daß der Ausgang des FSK-Demodulators direkt TTL-kompatibel ist.

Wenn die Anwendung von CMOS-Schaltungen im linearen Betrieb also zwar mit einigen Problemen verbunden ist und man sich genau überlegen muß, welche Gatter man unter welchen Bedingungen einsetzt, so zeigt die FSK-Demodulator-Schaltung doch, was man mit solchen ICs alles machen kann – vom Mikrofonverstärker bis zum selektiven Filter.

## Was meint die Post dazu?

Wie schon erwähnt, ist in Deutschland die elektrische oder akustische Ankopplung von Datenübertragungseinrichtungen nur nach einer Prüfung der Geräte im Fernmeldetechnischen Zentralamt gestattet. Allerdings ist es ziemlich aussichtslos, sich selbst ein Modem zu bauen und dann eine FTZ-Prüfung zu beantragen – nicht, weil man die technischen Anforderungen nicht einhalten könnte, sondern, weil man derzeit mit Wartezeiten in der Größenordnung von einem Jahr rechnen muß. Ferner muß eine Datenverbindung stets so aufgebaut sein, daß mindestens eine Seite mit einem post-eigenen (!), elektrisch angekoppelten Modem arbeitet. Schlechte Zeiten für Möchtegern-Datenübertrager... Fe.



# Stringmanipulationen – höchst flexibel

Die Überlegenheit der drei Microsoft-Basic-Dialekte Commodore-Basic, Tandy-Level-II und Applesoft gegenüber früheren Versionen von Basic beruht nicht zuletzt auf der beeindruckenden Eleganz, mit der die Zeichenketten-Verarbeitungsmöglichkeiten dieser Sprachen ausgestattet sind. Strings können in Test wie numerische Variable behandelt und auf „größer“, „kleiner“, „gleich“ und „ungleich“ getestet werden, wobei der jeweilige ASCII-Code der Zeichen in der Kette der Grundlage des Vergleichs bildet. Hierauf gehen die Handbücher der betroffenen Rechner PET, TRS-80 und Euro-Apple zwar in ausreichender Ausführlichkeit ein, nicht jedoch in der für Anfänger wünschenswerten Breite auf die Möglichkeiten zur Manipulation von Zeichen innerhalb der Kette. Lediglich vier Arten von Statements – eines davon liegt in zwei Versionen vor – erlauben nämlich, miteinander kombiniert, die Isolation eines Teilstrings aus einer Gesamtkette aufgrund beliebiger Anforderungen an das Programm. Während die Handbücher diese Grundoperationen noch vorstellen, verzichten sie auf eine Darstellung der Möglichkeiten, die sich durch Kombination dieser Fundamentalstatements und durch den Ersatz von Konstanten durch Variable ergeben. Im einzelnen sind diese Grundstatements:

1. Der Befehl RIGHTS, gefolgt von einer Klammer, in der – durch Komma getrennt – der Name des Ausgangs-Strings und die Zahl der zu isolierenden Zeichen anzugeben ist. RIGHTS isoliert den rechten Teilstring.
2. Der Befehl LEFTS (gleiche Syntax wie RIGHTS) isoliert den linken Teilstring.
3. Der Befehl MIDS. Syntaxvorschrift hier: Wiederum Klammer mit Stringnamen, sodann die Position des ersten zu isolierenden Zeichens, gefolgt von einem zweiten Komma und der Angabe der Zeichenanzahl, die man zu entnehmen wünscht. Wird auf die Angabe dieses dritten Parameters verzichtet, isoliert der Befehl bis zum Kettenende.
4. Schließlich noch die Funktion LEN, gefolgt von einem Stringnamen in Klammer; sie gibt die Anzahl der Zeichen im String zurück.

In Bild 1 sind am Beispiel eines Strings mit dem Inhalt „ALLER GUTEN DINGE SIND DREI“ die Verhältnisse näher demonstriert. Eine Besonderheit bringt die letzte Zeile: Hier ist als zweiter Parameter nämlich nicht länger eine Konstante enthalten, sondern die Differenz aus einem Funktionswert und einer Konstanten. Der Grund dafür: Die drei klassischen Microsoft-String-Zugriffe RIGHTS, MIDS und LEFTS erlauben in Verbindung mit Konstanten als Parameter wahlfreien Zugriff auf Teilmglieder der Zeichenkette nur bei konstanter Stringlänge. Bei der Aufgabe zum Beispiel, aus einer Zeichenkette beliebiger Länge die – von rechts nach links gezählt – Teilkette ab Zeichen 6 bis Zeichen 15 zu entnehmen, versagt MIDS, weil es ja – wie weiter oben demonstriert – als zweiter Parameter die Angabe der (von links nach rechts beziffert) Platznummer im String verlangt, von der ab zu isolieren sei. Hier hilft Ersatz der geforderten Konstanten durch einen Ausdruck mit der Funktion LEN weiter, wie das Beispiel zeigt.

Bis hierhin ist die hohe Nützlichkeit des Verzichts auf Konstanten bei den Parameter-Angaben noch nicht sehr

deutlich sichtbar geworden, weil zur Demonstration aus didaktischen Gründen ein nicht sehr praxisnahes Beispiel gewählt werden mußte. Dies wird aber sofort anders, wenn man einmal eine Aufgabe betrachtet, wie sie in der Praxis häufiger vorkommt. Dazu gehören die Suche nach einem Zeichen im String,

```

LIST
10 B = 0: PRINT : INPUT " PRUEFEK
   TIE BITTE? ":A$
20 L = LEN (A$): FOR N = 1 TO L
30 IF MID$(A$,N,1) = "*" THEN
   B = N: N = L
40 NEXT : PRINT : IF B = 0 THEN
   PRINT " +++ KEIN BEGRENZER
   " : GOTO 10
50 PRINT : PRINT " +++ BEGRENZER
   IN STELLE "B
60 PRINT "KETTE BIS BEGRENZER: "
   LEFT$(A$,B - 1)
70 PRINT "KETTE AB BEGRENZER: "
   RIGHT$(A$,L - B)
80 GOTO 10

JRUN
PRUEFEKETTE BITTE? ANTON+BERTA+CAESAR
+++ BEGRENZER IN STELLE 6
KETTE BIS BEGRENZER: ANTON
KETTE AB BEGRENZER: BERTA+CAESAR

PRUEFEKETTE BITTE? ANTON+BERTA+CAESAR
+++ BEGRENZER IN STELLE 12
KETTE BIS BEGRENZER: ANTON+BERTA
KETTE AB BEGRENZER: CAESAR
    
```

Bild 2. Die Stringbefehle lassen sich ideal mit String-Variablen verknüpfen

von dem nicht die Position, sondern allein der zugehörige ASCII-Code bekannt ist, sowie die anschließende Benutzung dieses Zeichens als Begrenzer. Listing und Probelauf-Ausdruck in Bild 2 zeigen, daß die Kombination Stringbefehl mit Variablen und Funktionen als Parameter in Microsoft-Sprachen diese Aufgabe mit weitaus geringerem Aufwand lösbar macht, als dies in älteren Basic-Dialekten möglich war. Die Schleife 20 bis 40 sucht selbsttätig den Begrenzer – hier das Sternchen-Zeichen –, und übergibt die Platznummer der gefundenen Position in der Variablen B, die in den Zeilen 60 und 70 so dann der Stringtrennung dient. Wunderschöne Möglichkeiten eröffnen sich hier – allerdings, die Handbücher schweigen darüber...

Hans-Georg Joepgen

```

JA$="ALLER GUTEN DINGE SIND DREI"

JPRINTLEFT$(A$,5)
ALLER

JPRINTRIGHT$(A$,4)
SIND DREI

JPRINTMID$(A$,10)
DINGE SIND DREI

JPRINTMID$(A$,10,5)
DINGE

JPRINTMID$(A$,LEN(A$)-14,10)
DINGE SIND
    
```

Bild 1. Abtrennen von Stringteilen mit LEFTS, RIGHTS und MIDS





## Ein Netzteil für den AIM-65

Der Mikrocomputer AIM-65 von Rockwell (bzw. der PC-100-KIT von Siemens) benötigt zwei Versorgungsspannungen: 5 V und 24 V. Das im Bild gezeigte Netzteil ist ideal für die Stromversorgung des AIM-65 geeignet und sehr einfach aufzubauen, z. B. mit einem kleinen Metallgehäuse und einer Lötösenleiste.

Die 5-V-Stabilisation übernimmt der Leistungsregler LM 323 (Preis ca. 10...20 DM), während die 24-V-Versorgung für den AIM-Thermodrucker nicht stabilisiert wird. Ein 0,47- $\Omega$ -Widerstand vor dem Ladeelko des 5-V-Reglers verbessert den Stromflußwinkel im Gleichrichter und damit die Trafoausnutzung; ferner vermindert er Impulsstörungen im Netz.

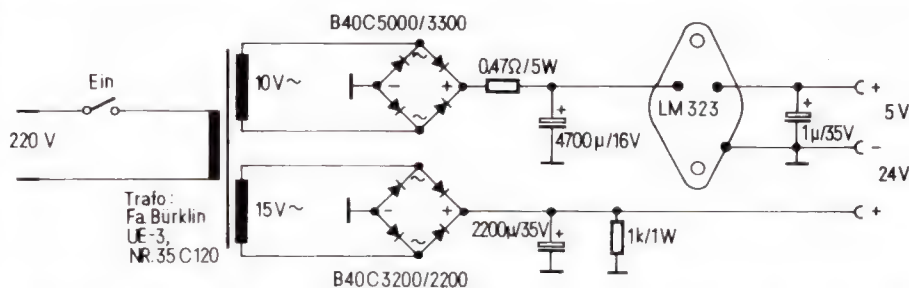
Das Netzteil kann bei 5 V rund 2 A liefern, bei 24 V rund 1 A (wobei durch die Größe des Ladeelkos die höheren Impulsströme des Druckers leicht verkraftet werden).

Der LM 323 ist nicht gerade billig; eine Alternative ist das Parallelschalten zweier LM 309, was keinerlei Nachteile mit sich bringt. Welcher Regler auch verwendet wird: Auf eine ausreichende Kühlung ist unbedingt zu achten, ein Fingerkühlkörper ist kaum geeignet. Besser ist ein Rippen-Kühlkörper mit geringem Wärmewiderstand, d. h. ei-

ner Grundfläche von mindestens etwa 5 cm x 10 cm und Bohrungen für das TO-3-Gehäuse des Reglers.

Der 1-k $\Omega$ -Widerstand parallel zur 24-V-Versorgung verhindert beim Abschalten des Netzteils einen unnötigen Papierausstöß des Druckers, der leer läuft, wenn die 5-V-Versorgung fehlt.

Herwig Feichtinger



Mit wenig Materialaufwand lassen sich die beiden Versorgungsspannungen für den AIM-65 oder PC-100-KIT bereitstellen. Auf eine ausreichende Kühlung des Reglers LM 323 ist unbedingt zu achten

## The Basic Handbook

Der Computer-Hobbyist und Dekan an einem amerikanischen College, David A. Lien, begleitet seit Jahren durch ungewöhnlich sorgsam vorbereitete Veröffentlichungen das Geschehen in der Mikrocomputer-Szene. Zugänglich sind seine Bücher und Artikel nahezu ausschließlich leider bis zur Stunde allein englischsprachigen Fachliteratur-Lesern; in Deutschland kennt man Lien bisher hauptsächlich durch die TRS-80-Handbücher, die aus seiner Feder stammen.

Mit seinem jüngsten Werk nun hat Lien sich einer Aufgabe angenommen, deren Erledigung schon längst fällig war, nämlich einmal etwas Ordnung zu bringen in die auf unterdessen rund 60

angewachsenen Basic-Dialekte, die untereinander nur zum Teil kompatibel sind. Zwar hat das American National Standards Institute, das US-Äquivalent zu den deutschen DIN-Gremien, unterdessen ein „ANSI BASIC“ definiert, und amerikanische Hardware- und Software-Produzenten, die ihre Hochsprachen als „Basic“ bezeichnen, müssen den Anforderungen der ANSI-Norm nachkommen; aber niemand hindert sie daran, ANSI-Basic durch „hausgestrickte“ Statements zu ergänzen.

So kommt es, daß heutzutage trotz aller Normung eine schier babylonische Sprachverwirrung zwischen den unterschiedlichsten Basic-Dialekten

herrscht; dies geht so weit, daß für ein und dasselbe Basic-Wort in verschiedenen Basic-Versionen vollkommen verschiedene Bedeutungen gelten.

Lien hat sich nun die Mühe gemacht, in enzyklopädischer Form die Eigenheiten einer großen Zahl von Basic-Abarten einander gegenüberzustellen und dabei stellenweise sogar zu erläutern, wie man Spezial-Funktionen, die im eigenen Interpreter nicht zur Verfügung stehen, durch Software simulieren kann.

(David A. Lien, „The Basic Handbook“, Compusoft Publishing, P.O. Box 19669, San Diego, California, USA; 14.95 Dollar.) Hans-Georg Joepgen



# Das VIA 6522 -

## ein intelligenter Interfacebaustein

### Was ist Interfacing?

In blockmäßiger Beschreibung besteht ein Mikroprozessor aus Zentraleinheit (CPU) mit Taktgenerator, Speichern und Eingabe/Ausgabe (E/A oder I/O). Als Festwertspeicher für Programme, Konstante usw. dienen dabei ROMs, PROMs, EPROMs, Zwischenergebnisse werden in RAMs als Schreib-/Lesespeicher abgelegt. Allein mit CPU und Speichern könnte ein Prozessor zwar arbeiten, er könnte aber von der Umwelt keine Signale und Anweisungen empfangen und auch keinerlei Ergebnisse und Steuerungen dorthin abgeben. Daher der Bedarf für Schnittstellen, Interfaces zur Ein- und Ausgabe. Schnittstellen mögen dabei zu anderen elektronischen Systemen z. B. zur Prozeßsteuerung bestehen oder zu Geräten, die der Verständigung mit dem Menschen dienen, wie Tastaturen, Anzeigen, Drucker. Für das Interfacing wurde eine Vielzahl von Bausteinen geschaffen, die oft nur für eine bestimmte Anwendung ausgelegt sind, man denke an Tastaturenkoder oder Video-Controller.

Die Prozessorbausteine CPU, Speicher und E/A werden untereinander durch drei Leiterbahnsysteme verbunden, den Adreßbus, den Datenbus und den Steuerbus (s. Bild 1).

Die Busse werden von der Zentraleinheit regiert. Auf dem im allgemeinen 16 bit breiten Adreßbus sendet sie Adressen zur gezielten Ansprache einzelner Speicherzellen, der Datenbus (8 bit) transportiert Informationen in zwei Richtungen, entweder zur CPU oder

von der CPU in die Speicher. Der Kontrollbus schließlich führt Signale zur Synchronisation aller Bauteile und für die Bestimmung, ob es sich um einen Schreib- oder Lesezyklus der Zentraleinheit handelt.

Für die sehr verbreitete 65XX-Prozessorfamilie, repräsentiert durch die Systeme AIM-65, PC-100, PET, SYM-1 und KIM-1, besprechen wir den außerordentlich leistungsfähigen Versatile Interface Adapter (VIA) 6522 (Bild 2). Er wird von Rockwell, MOS-Technology und Synertek hergestellt und ist auf den vier erstgenannten Geräten z. T. mehrfach enthalten, um die Systemhardware zu bedienen. Aber er steht dem Benutzer dort auch für eigene Anwendungen zur Verfügung (Application Connector, bzw. User Port).

Dieser Interfacebaustein eignet ebenso für andere Prozessoren, vor allem für die sehr verwandte 68XX-Serie von Motorola. Er weist große Ähnlichkeiten mit dem 6820/6520-PIA auf (Peripheral Interface Adapter) und auch mit dem IC 6530 im KIM-1. Das VIA vermag aber wesentlich mehr als die eben genannten. – Unsere Darstellung soll die bei den Herstellern und ihren Distributoren erhältlichen Datenblätter nicht ersetzen, sondern die vielen Möglichkeiten des Interfacing mit dem VIA in einer Übersicht aufzeichnen.

In der blockmäßigen Vereinfachung gemäß Bild 2 können wir das VIA als einen mit der CPU über Adreß-, Daten- und Steuerbus verbundenen Interfacebaustein mit 16 Registeradressen bezeichnen, der zur Verbindung mit der

Umwelt 2 Ports mit je 8 E/A-Pins und 4 Pins für Steuerfunktionen trägt. Insgesamt stehen damit 20 E/A-Leitungen zur Verfügung.

### Verbindung zur CPU und Adressierung

Im Konzept der 65er-Familie werden die Register der Interfacebausteine von der CPU wie ganz normale Speicherzellen gelesen und beschrieben. Daher der volle Anschluß an den Datenbus und an die Steuerleitungen R/W und  $\Phi 2$ . Um Pins zu sparen, werden nicht alle 16 Adreßleitungen auf den Chip gelegt, sondern nur 4 als Register Select (RS0-RS3). Das reicht zur Ansprache der 16 Maschinenregister aus. Und über eine externe Decodierlogik wird aus den Signalen der höheren Adreßbusleitungen ein Chip-Select gebildet (CS1, CS2).

Je nach dem bei der Decodierung betriebenen Hardwareaufwand mag sich dabei für mehrere Interfacebausteine eine lückenlose Aneinanderreihung von Registeradressen ergeben oder auch nicht. Im letzteren Fall mag ein Baustein wegen zu grober Decodierung die Adressen einer ganzen Page oder gar eines ganzen KByte belegen. Für eine spätere Expansion des Systems im vorhandenen Adressenraum kann das hinderlich sein.

Zur Verbindung mit dem Steuerbus gehört auch die Interrupt-Leitung IRQ. Damit ist nicht gesagt, daß das VIA im Interrupt betrieben wird. Vielmehr: Es darf einen Interrupt auslösen, wenn der Programmierer das für gewisse Ereignisse vorgesehen hat.

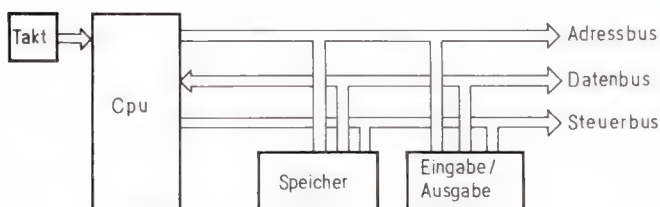


Bild 1. Blockschaltbild eines einfachen Mikrocomputers

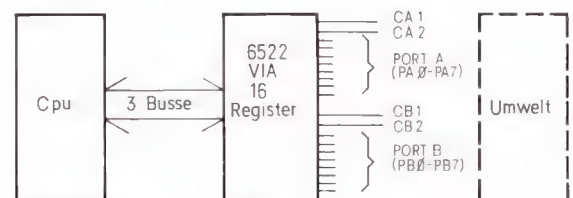


Bild 2. Der Mikroprozessor kann über das VIA 6522 mit der Außenwelt kommunizieren



## Funktionskontrolle durch 16 Register

Die modernen Interfacebausteine enthalten ein feines Netz digitaler Schaltungen. Sie können nach dem Willen des Anwenders in ihrer Funktion programmiert werden, indem er bei Programmbeginn gewisse Bitmuster in die Steuerregister des Interfacebausteines hineinschreibt. Dadurch werden die benötigten digitalen Funktionen auf dem Baustein softwaremäßig durchgeschaltet. Seitens der CPU erfolgt diese Initialisierung mit STA-Befehlen (speichere Akku). Mit der Zahl der jeweiligen Register steigt im allgemeinen auch die Vielseitigkeit eines Peripheriebausteines. Dem VIA sind 16 Registeradressen zugeordnet, in Wirklichkeit ist eine noch größere Zahl an Steuerungen im Einsatz.

Auf die Register des VIA kommen wir noch im einzelnen zu sprechen. Wir gliedern sie zunächst schematisch wie folgt:

Zwei Datenrichtungsregister, DDRA und DDRB, die die Funktionen der Ports bitweise (pinweise) für Eingabe oder Ausgabe festlegen.

Zwei Ports mit Ausgaberegister ORA, bzw. ORB, Leseregister IRA (IRB) und einschaltbarem Lese-Latch.

Zwei 16 bit breite Timer, die zahlreiche Funktionen ausüben können, wie Impulsweitenmessung, Ereigniszählung, Erzeugung von einmaligen oder stetigen Rechteckimpulsen an PB7 und Kontrolle seriellen Datenverkehrs.

Schieberegister SR für die serielle Ein- und Ausgabe an Pin CB2.

Hilfsregister ACR, das die Funktionen der Timer, der Pins PB7 und PB6 und die Vorspeicherung einkommender Signale an den Latches der Ports A und B reguliert.

Steuerregister PCR zur Funktionskontrolle der Pins CA1, CA2, CB1 und CB2 für die Reaktion auf Impulsflanken und für den Handshake-Verkehr mit Peripherieeinheiten.

Interrupt-Anzeigeregister IFR. Die Interruptkontrolle findet in 3 Stufen statt. Im IFR können die möglichen Interrupt-Quellen (nämlich die 4 Steuerleitungen, die beiden Timer bei Nulldurchgang und das Schieberegister nach Abarbeitung) lediglich ein Flag setzen. Bit 7 ist logisch ODER auf die nachfolgenden Flags gesetzt und wird

Tabelle: VIA-Registeradressen beim AIM 65 und beim PET2001

AIM Anwender-VIA hexadezimal	AIM System-VIA hexadezimal	PET2001 System- und Anwender-VIA hexadezimal bzw. dezimal	Register
A000	A800	E840 59456	ORB Ausgaberegister Port B
A001	A801	E841 59457	ORA Ausgaberegister Port A (mit Handshake)
A002	A802	E842 59458	DDRB Datenrichtungsregister für Port B
A003	A803	E843 59459	DDRA Datenrichtungsregister für Port A
A004	A804	E844 59460	Timer 1: T1C-L Lesen Zähler .low', Schreiben Vorspeicher .low'
A005	A805	E845 59461	T1C-H Lesen und Schreiben Zähler .high'
A006	A806	E846 59462	T1L-L Lesen oder Schreiben in Vorspeicher .low'
A007	A807	E847 59463	T1L-H Lesen oder Schreiben in Vorspeicher .high'
A008	A808	E848 59464	Timer 2: T2C-L Lesen Zähler .low', Schreiben Vorspeicher .low'
A009	A809	E849 59465	T2C-H Lesen und Schreiben Zähler .high'
A00A	A80A	E84A 59466	SR Schieberegister
A00B	A80B	E84B 59467	ACR Hilfsregister
A00C	A80C	E84C 59468	PCR Steuerregister
A00D	A80D	E84D 59469	IFR Interrupt-Anzeigeregister
A00E	A80E	E84E 59470	IER Interrupt Enable Register
A00F	A80F	E84F 59471	ORA Ausgaberegister für Port A (ohne Einfluß auf Handshakesignal)

,1', sobald nur ein Flag ,1' wird. Das ermöglicht eine schnelle Abfrage mit dem BIT-Befehl.

Interrupt Enable Register IER. Für jedes der vorgenannten Flags (Interruptquellen) kann durch Setzen seines Bits in diesem Register bestimmt werden, ob es einen Interrupt auslösen darf. Wenn ja, dann bringt das auslösende Ereignis das Ausgangssignal IRQ des Chips auf ,0' (dritte Stufe).

In der Tabelle sind für den AIM-65 die Adressen des dem Anwender voll zur Verfügung stehenden ,User VIA' und des für Systemzwecke eingesetzten VIA aufgeführt. für den PET die Adressen des gemischt genutzten VIA. Die Hälfte der Leitungen ist hier für das System genutzt, die 8 Leitungen des Port A sowie CA1 und CB2 sind auf den ,User Port" geführt.

## Die Ports als E/A-Leitungen

Die wohl einfachste Nutzung des VIA besteht darin, an den Ports Signale zu senden oder zu empfangen. Ein Port muß dabei keineswegs nur als Eingang oder nur als Ausgang betrieben werden, er kann beide Funktionen gemischt wahrnehmen, weil jeder Pin individuell geschaltet werden kann. Dazu schreibt man zunächst ein Bitmuster in das Datenrichtungsregister DDRA bzw. DDRB. Eine ,1' schaltet den dazugehörigen Pin auf Ausgang, eine ,0' auf Eingang. Das eigentliche Ausgangssignal wird erst danach durch Einschreiben in das Ausgangsregister ORA (ORB) erzeugt. Für das Beispiel gemäß Bild 3, das die Pins PB7-5 als Eingang schaltet und PB4-0 als Ausgang, programmiert man:

```
LDA #$1F    FESTLEGUNG
             DER PINS FÜR E/A
STA DDRB    IM DATEN-
             RICHTUNGSREGISTER
LDA #$1A    ERWÜNSCHTE AUSGANGS-
             SIGNALE „HIGH“ UND „LOW“
STA ORB     IN DEN PORT
```

Über die TTL-Kompatibilität und Belastbarkeit der Pins orientiere man sich in den Datenblättern. Daß PB7 als Output für Timer-kontrollierte Impulse

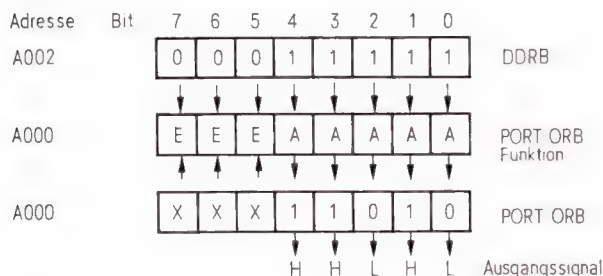


Bild 3. Ein Port besteht aus acht Leitungen; ob es sich dabei um Ein- oder Ausgänge handelt, wird per Software mit dem Datenrichtungs-Register DDRB (hier Port B) festgelegt



Sonderfunktionen übernehmen und daß PB6 für Ereigniszählung dienen kann, wurde bereits erwähnt und findet weitere eigene Darstellung.

Während die Dinge für das Schreiben in die Peripherie übersichtlich und einfach liegen, sind für das Lesen einige Besonderheiten zu beachten, mit deutlichen Unterschieden für die A- und B-Seite des VIA. Zunächst ist zu erwähnen, daß jede Seite neben dem Output-Register (ORA oder ORB) zwei weitere Schaltungen aufweist, das Leseregister (Input Register IRA bzw. IRB) und das Lese-Latch. Was in diesem Leseregister steht und was damit auf den Datenbus gelangt, hängt nicht nur vom anstehenden Signal, sondern auch von der Funktion ab, die den Steuerleitungen zugewiesen wurde. Vorgreifend ist darauf hinzuweisen, daß die Steuerleitungen CA1 und CB1 durch Einschreiben in das Steuerleitungsregister PCR auf das Erkennen aktiver Übergänge geschaltet werden können (aufsteigende oder abfallende Impulsflanke) und daß in Abhängigkeit davon verschiedene Vorgänge im Chip auslösbar sind, unter anderem auch ein Handshake an dem korrespondierenden Pin CA2 bzw. CB2 als Ausgang (Handshake Control).

Und dann gibt es da noch die Latching-Kontrolle, von den beiden niederwertigsten Bit des Hilfsregisters ACR kontrolliert und unabhängig für die A- und für die B-Seite ausübbar. Das bedeutet: Man liest den Port so, wie er sich im Moment des Lesens befindet (das ACR-Bit ist in Normalstellung auf „0“) oder so, wie er sich vor dem aktiven Übergang am CA1- bzw. CB1-Pin befunden hatte (ACR-Bit „1“).

Damit nicht genug der Besonderheiten. Es sind ja Fälle denkbar, in denen ein Ausgangspin unter der zu treibenden Last unter die Spannungsschwelle absinkt, die mit „high“ erkannt wird, obwohl er mit „high“ beschrieben wurde. Im Leseregister B lesen wir in diesen Fällen einen Mix aus Pin-Ausgangssollsignalen und tatsächlichen Eingangspegeln, solange das von CB1 gesetzte Interrupt-Flag „high“ ist. Sobald dieses Flag gelöscht ist, liest man die tatsächlichen Pegel an den Pins. „low“ und „high“.

Nachzutragen ist: Für den Prozessor haben die Leseregister IRA, bzw. IRB dieselben Adressen wie die Ausgangsports. Sofern man nicht im Handshake-Betrieb arbeitet (s.u.), kann man die A-Seite gleichmäßig in den Adressen A001 und A00F auslesen. Das Latching eröffnet die Möglichkeit, die Eingangskombination nach dem aktiven Übergang mit derjenigen zuvor z. B. mit logisch EOR zu vergleichen. Dabei werden die Veränderungen erkannt.

Das nachfolgende Beispiel initialisiert wie folgt: CB1 reagiert auf aufsteigende Impulsflanke, Port B ist als Eingang geschaltet mit Latching der Signale.

```
LDA #502    ERMÖGLICHE LATCHING
             AN DER B-SEITE
STA ACR     HILFSREGISTER
LDA #510    CB1 AUF AUFSTEIGENDE
             FLANKE TRIGGERN
STA PCR     STEUERREGISTER
LDA #500    PORT B ALS INPUT
STA DDRB    DATENRICHTUNGS-
             REGISTER
```

Bei dieser Initialisierung wird ein aktiver Übergang an CB1 das Interrupt-Flag dieses Pins setzen und zugleich auch logisch ODER das Bit 7 im Interrupt-Anzeigeregister. Im weiteren Programmverlauf kann man daher die Interruptbedingung wie folgt abfragen:

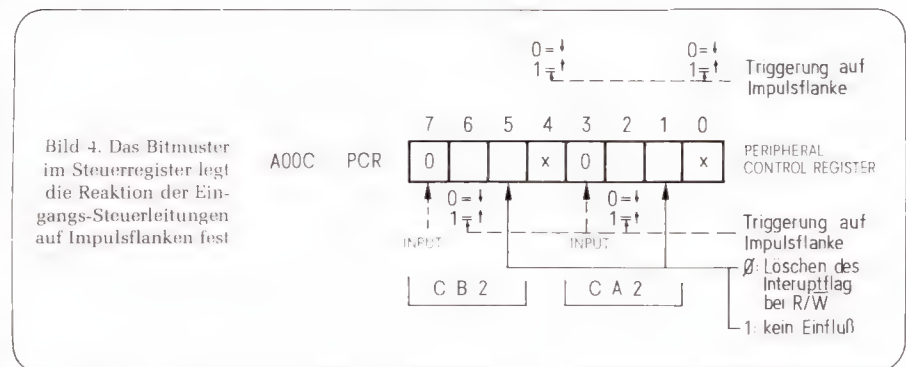
```
LDA #510    MASKE
W BIT IFR    DER BIT-BEFEHL VOLLZIEHT
             LOGISCH AND MIT DER
             MASKE
BEQ W        WARTESCHLEIFE
```

Die häufig gebrauchten Techniken der Portbedienung dürften weitgehend bekannt sein. Z. B. Abfrage, ob sich irgendein Eingangssignal geändert hat:

Durch Beschaltung eines Ports mit z. B. 2 Stück 4 zu 16 Decodern/Multiplexern kann die Zahl der möglichen Verbindungen zur Außenwelt ggfs. vervielfacht werden. Es sind natürlich auch andere Beschaltungen für das Multiplexing möglich, wie z. B. mit dem 4016 oder 74 157. Mit einem Steuerbit könnte man 2 x 7 Datenbits umschalten.

## Impulsflanken an den Eingangs-Steuerleitungen

Die vier Pins CA1, CA2, CB1 und CB2 können im Steuerleitungsregister PCR einzeln so programmiert werden, daß sie als Eingangsleitungen und in Reaktion auf steigende oder fallende Impulsflanken ihr korrespondierendes Interrupt-Flag im Interrupt-Anzeigeregister auf „1“ setzen. Ob daraus nur eine Abfragemöglichkeit oder ein tatsächlicher Interrupt wird, hängt von der Programmierung des Interrupt-Enable-Registers IER ab. Wie in Bild 4 dargestellt, bestimmen die Bitpositionen 6, 4, 2 und 0 im PCR für CB2, CB1, CA2 bzw. CA1, ob die Reaktion bei fallender (= 0) oder bei steigender (= 1) Impulsflanke erfolgt.



```
LDA PORT    ALTE SIGNALKOMBINATION
W CMP       WARTE, BIS SICH ETWAS
PORT        ÄNDERT
BEQ W       WARTESCHLEIFE
```

An Ausgangsports lassen sich – unabhängig von den besonderen Fähigkeiten des VIA – Rechteckimpulse durch Flippen des niedrigsten Bits wie folgt erzeugen:

```
INC PORT    BIT 0 AUF „1“ SETZEN,
             WENN ES ZUVOR „0“ WAR
DEC PORT    UND WIEDER AUF „0“
```

Für Tastaturabfragen u. ä. ist es geläufig, eine Rasterung von Gitterpunkten in der Weise vorzunehmen, daß man eine logische „0“ durch einen Sendeport shiftet (ASL oder LSR) und an einem Empfangsport abfragt, in welcher Bitposition die „0“ empfangen wurde. Man nimmt die „0“, weil unbeschaltete offene Leiterbahnen „1“ senden.

CA1 und CB1 können bei aktivem Übergang am Eingang nur ihr Interrupt-Flag setzen. CA2 und CB2 werden durch Bit 3 = „0“ im PCR bzw. Bit 7 = „0“ als Eingänge geschaltet. Und die Bitposition 1 bzw. 5 bestimmt, ob das Interrupt-Flag durch Lesen oder Schreiben des Ports automatisch gelöscht wird („0“: automatisch, „1“: kein Einfluß).

Beispiel: CB2 soll das Input bei aufsteigender Impulsflanke Interrupt auslösen. Ein Lesen/Schreiben des Port B soll das Interrupt-Flag automatisch löschen:

```
LDA #540    CB2-KONTROLLE
STA PCR     STEUERLEITUNGS-
             REGISTER
LDA #588    INTERRUPTFREIGABE FÜR
             CB2. WEGEN BIT 7 =
             „1“ S. U.
STA IER
```



## Die Steuerleitungen als Ausgang und Handshaking

Handshaking ist beim asynchronen Verkehr zwischen Prozessor und Peripherieeinheit der Signalaustausch zum Zwecke der zuverlässigen Synchronisation von Eingabe oder Ausgabe über die Ports. Englisch spricht man von ‚talker‘ und ‚listener‘ für die betroffenen Einheiten. Vom Talker wird ein Signal ‚data ready‘ und vom Listener eines ‚data taken‘ erwartet. Beim 6522 VIA stehen für diesen Signalaustausch die bereits genannten 4 Steuerleitungen zur Verfügung.

CA1 und CB1 dienen als Input für die Erkennung aktiver Übergänge. Sie setzen Interrupt-Flags. Ist eine sendende Peripherieeinheit angeschlossen, so bedeutet ihr Signal an den Eingangspins ‚ready to transmit to processor‘. Handelt es sich dagegen um eine Ausgabeeinheit, so signalisiert sie dem Prozessor auf diesen Leitungen ‚ready to receive‘ oder ‚data taken‘.

Im Handshake-Verkehr dienen dann CA2 oder CB2 als Ausgabeleitungen, um einer sendenden Einheit mitzuteilen ‚data taken‘ oder um einer Ausgabeeinheit anzukündigen ‚data ready for transmission‘.

CA1 (CB1) ‚data taken‘ gemeldet wird. Erst jetzt wird das Interrupt-Flag gesetzt, damit der Prozessor neue Daten am Port bereitstellen kann.

Beispiel: Am Port A sollen Daten im Interrupt-Betrieb empfangen werden. Die sendende Einheit meldet mit aufsteigender Flanke an CA1 ‚data ready‘ und will einen kurzen Impuls über CA2 für ‚data taken‘ erhalten:

```
LDA #$0B  CA1 AUF AUFSTIEGENDE FLANKE,
           CA2 AUF PULSAUSGABE
STA PCR   STEUERLEITUNGSREGISTER
LDA #$82  ERMÖGLICHE INTERRUPT FÜR CA1
STA IER
```

Unabhängig von solchem Handshake-Betrieb können CA2 und CB2 unter Kontrolle des Programmes auch auf definierte Pegel ‚high‘ oder ‚low‘ gesetzt werden (manual output mode) um z. B. äußere Multiplexer umzusteuern.

Die Vielseitigkeit der Ausgangsteuerung läßt für die Bedienung wohl kaum Wünsche offen, zumal noch eine Ergänzung hinzukommt: Im handshake mode an Port A geht CA2 auf ‚low‘, wenn man den Port in Adresse A001 liest. Wenn neue Daten noch nicht bereitstehen, könnte das in zeitlicher Überschneidung zu Mißverständnissen

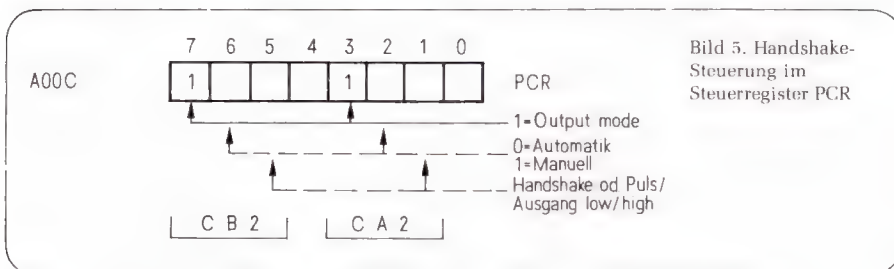


Bild 5. Handshake-Steuerung im Steuerregister PCR

CA2 und CB2 werden als Ausgänge geschaltet, indem man in Bitposition 3 bzw. 7 des PCR eine ‚1‘ einschreibt (Bild 5). Für jede Seite des VIA gibt es 4 Möglichkeiten zur Erzeugung des Ausgangssignales. Die eleganteste finden wir nur auf der A-Seite, das automatische Handshaking: Sobald man (bei der Eingabe in den Prozessor) in der Prüf- oder Interruptroutine den Port A liest, geht CA2 in automatischer Beantwortung auf ‚low‘. Das bedeutet ‚data taken‘. Im handshake mode geht CA2 erst dann wieder auf ‚high‘, wenn die Peripherie durch aktiven Übergang an CA1 meldet ‚new data ready‘. Daneben gibt es auch eine Betriebsart, in der CA2 oder CB2 mit einem kurzen Impuls quittieren.

Ähnlich sinnvoll liegen die Verhältnisse bei der Ausgabe an die Peripherie: Ein Beschreiben des Ports A oder B setzt CA2 bzw. CB2 auf ‚low‘. Damit wird ‚data ready for transmission‘ signalisiert. Das Ausgangssignal verharrt auf diesem Pegel, bis von draußen über

führen. Es wurde daher für Zwischenprüfungen die Adresse A00F geschaffen, deren Einlesung oder Beschreibung die Handshake-Steuerung nicht verändert. Damit ist auch die Option eröffnet, den Port gemischt als Ein- und Ausgang zu fahren. Die Steuerung entsprechender Pins eines Ausgangs oder die Prüfung eines zweiten Einganges beeinflusst dann nicht die ebenfalls angeschlossene Handshake-Einheit.

## Die Timer/Zähler

Das VIA trägt zwei Timer, T1 und T2, von je 16 bit Breite. Anders als bei den Timern z. B. des 6532 RIOT und des 6530 RRIOT (im KIM) arbeiten sie nicht mit einstellbaren Teil- oder Zählgeschwindigkeiten (z. B.: 8, : 1024), sondern zählen die 2 Byte breite Vorgabe mit jedem Maschinentakt um 1 herunter. Beide Timer setzen bei ihrem Nulldurchgang ein eigenes Interrupt-Flag im Interrupt-Anzeigeregister (IFR) und können auch wahlweise einen Interrupt auslösen, und zwar unter Kontrolle des im Interrupt Enable Register (IER) hinterlegten Bitmusters.

Den Timern ist gemeinsam, daß sie für die CPU definierte Zeit- und Verzögerungsintervalle erzeugen können. Ihre eigentliche Leistungsfähigkeit geht weit darüber hinaus. Timer T1 kann man etwa als Erzeuger von vielfältigen Rechteckimpulsen an Pin PB7 charakterisieren, T2 als Intervalltimer, Zähler und Kontrolleur für das serielle Schieberegister. Die Sonderfunktionen der Timer und der Pins PB7 und PB6 werden dabei vom Hilfsregister ACR kontrolliert.

Den Timern ist weiterhin die Zuordnung von Vorspeicher-Latches gemeinsam. T1 hat Latches für beide Bytes, T2 nur ein Latch für das niederwertige Byte.

### Der Timer T1

Diesem Timer sind 4 Registeradressen zugeordnet, bei dem Schreiben und Lesen teilweise jedoch eine unterschiedliche Bedeutung haben (Bild 6).

Zunächst ist festzuhalten, daß man in den niedrigen Zählerteil nicht direkt hineinschreiben kann, sondern nur in seinen Vorspeicher T1L-L. Das Laden dieses Zählerteils aus dem Vorspeicher erfolgt erst in dem Augenblick, da man in das hohe Zählerbyte, Adresse A005, hineinschreibt. Damit beginnt der ‚count down‘ gleichzeitig an der gesamten 16-bit-Vorgabe. Nach Nulldurchgang des Timers wird sein Interrupt-Flag gesetzt. Alle hier besprochenen Funktionen des Timers 1 werden durch die Bitposition 7 und 6 im Hilfsregister ACR bestimmt.

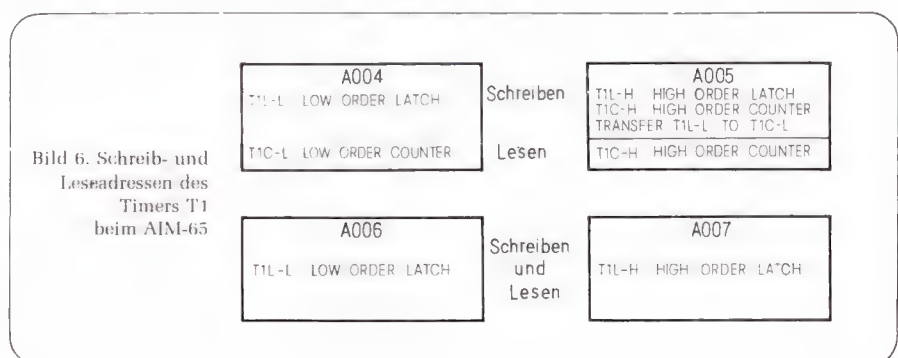


Bild 6. Schreib- und Leseadressen des Timers T1 beim AIM-65



Ein Beispiel für einfache Zeitverzögerung (one shot) von 160 µs, der Timer durchläuft nur einmal die Zählervorgabe. Interrupt wird durch Schreiben in das IER zugelassen:

```
LDA #$80    EINFACHES TIME OUT
STA ACR     HILFSREGISTER
LDA #$C0    INTERRUPTMÖGLICHKEIT FÜR T1
STA IER     INTERRUPT ENABLE REGISTER
LDA #$A0    TIMERVORGABE LOW, 160 µSEC
STA T1L-L   INS LATCH
LDA #$00    VORGABE HIGH
STA T1C-H   STARTE TIMER DURCH SCHREIBEN IN DEN ZÄHLER (A005)
```

Nach seinem Nulldurchgang wird der Timer weiter mit dem Systemtakt 02 heruntergezählt, so daß man ggfs. die Zeit seit dem Nulldurchgang erfassen kann. Das Interrupt-Flag wird dadurch gelöscht, daß man den Timer in T1C-L (A004) liest oder in T1C-H (A005) neu beschreibt.

Neben diesem einfachen time-out gibt es weitere drei Betriebsweisen. Da ist zunächst 'free running' zu erwähnen, Bit 6 im ACR ist gesetzt: Nach jedem time-out wird der Zähler automatisch mit dem Inhalt der Vorspeicher T1L-L und T1L-H (A006, A007) nachgeladen. Das Interrupt-Flag muß man dann aber per Programm zurücksetzen, indem man z. B. T1C-L liest. Damit sind wir auf eine weitere Besonderheit gestoßen: Der niedrige Zählerteil kann jederzeit gelesen werden, geladen wird er aber immer aus dem Vorspeicher-Latch. Der hohe Zähleranteil kann direkt gelesen und beschrieben werden, er wird automatisch aus dem Latch T1L-H nachgeladen, wenn mit ACR6 = '1' 'free running mode' gesetzt ist. Und damit wird auch gleich der Nutzen der Vorspeicher ersichtlich:

Bei 'free running' kann man den Timer irgendwann zwischenzeitlich mit der Zeitkonstanten für die nächste Phase vorladen, während er noch an der alten Phase herunterzählt. Nimmt man gar keine Veränderung der Werte in den Vorspeichern vor, so werden die Zeitabschnitte gleich lang, und es liegt eine Form der Selbstverwaltung im VIA vor. In nachfolgendem Beispiel nehmen wir einmal an, der Interruptvektor sei bereits geladen, und die Zeitabschnitte sollten alle gleich sein. Wir programmieren dann:

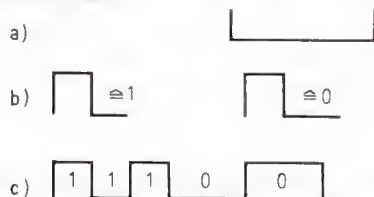


Bild 7. Impulserzeugung mit dem Timer T1 am Anschluß PB7: a) Einzel-Impuls (one-shot). b) Codierung von 1 oder 0 mit Startimpuls. c) Wechselphasencodierung

```
LDA #$40    FREE RUNNING
STA ACR     HILFSREGISTER
LDA #$XX    LADE NIEDRIGEN VORSPEICHER
STA T1L-L   INS LATCH
LDA #$YY    VORGABE FÜR DAS HÖHERE BYTE
STA T1C-H   STARTE TIMER UND LADE ZUGLEICH LATCH T1L-H
```

In der Interrupt-Routine setzen wir das Interrupt-Flag durch LDA T1C-L zurück.

Die beiden vorgenannten Betriebsweisen, die nur das Interrupt-Flag beeinflussen, werden ergänzt durch das 'PB7-Enable' (Bit 7 im ACR = '1'). PB7 dient jetzt als Ausgang für Rechteckimpulse, die von T1 generiert werden (Bild 7).

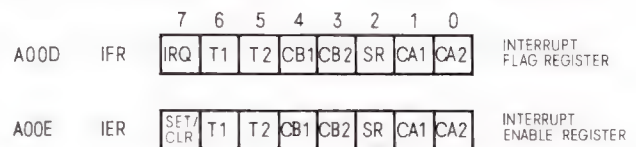
### Der Timer T2

Diesem Timer sind zwei Registeradressen zugeordnet, die wie beim T1 beim Lesen und Schreiben eine unterschiedliche Bedeutung für das niedrige Byte haben (Bild 8).

Bild 8. Schreib- und Leseadressen des Timers T2 beim AIM-65



Bild 9. Funktionen des Interrupt-Flag- und des Interrupt-Enable-Registers: die Adressen gelten wieder für den AIM-65



Auch hier kann der untere Zählerteil nicht direkt beschrieben werden, er wird in dem Moment aus seinem Vorspeicher T2L-L geladen, wo man in das obere Zählerbyte T2C-H einschreibt. T2 arbeitet im 'one shot mode' als Intervalltimer, wenn Bit 5 im ACR = '0'. Bei seinem Nulldurchgang setzt er einmalig sein Interrupt-Flag und zählt dann weiter von 'FFFF' herunter. Das Interrupt-Flag wird durch Lesen von T2C-L oder neues Beschreiben von T2C-H gelöscht.

Setzt man Bit 5 im ACR auf '1', so fungiert T2 als 'Count-Down-Zähler' für an PB6 angelegte abfallende Impulsflanken. Der Zähler ist mit einer Vorgabe zu laden. Das Interrupt-Flag erscheint, wenn die Vorgabe mit Nulldurchgang des Zählers verbraucht ist.

T2 kann unter gewissen Bedingungen auch als Taktgenerator für das Schieberegister SR eingesetzt werden (s. u.), dann ist allerdings nur der untere Zählerteil in Aktion.

### Das Schieberegister

Es dient dem seriellen Datenverkehr über Pin CB2 als Ein- oder Ausgabe. Intern besteht eine Verkoppelung mit einem Modulo-8-Zähler. Auf dem VIA sind für die Eingabe drei und für die Ausgabe vier Betriebsarten vorgesehen. Bei der seriellen Datenübertragung

kommt es wesentlich auf ein definiertes Timing zwischen Sender und Empfänger an. Pin CB1 übernimmt dabei neue Funktionen, die vom ACR gesteuert werden. Man kann dort den Takt einer externen Quelle anlegen (z. B. des Senders) oder einen intern erzeugten Takt für einen Empfänger über CB1 aussenden.

Als Takt an CB1 stehen zur Verfügung: a.) die Systemuhr 02, deren effektiver Takt mit 0,5 MHz arbeitet, b.) die Zeitvorgabe im niedrigen Teil des Timers 2, c.) z. B. Clockimpulse, die man im Timer 1 erzeugt und aus PB7 auf CB1 führt.

Hinsichtlich des Taktes bei serieller Datenübertragung beachte man in den Diagrammen des 6522-Datenblattes sehr genau, wann mit aufsteigender und wann mit abfallender Impulsflanke Daten übernommen werden. Im übrigen kann CB2 als Ausgabe für den Ruhezustand im PCR sowohl für 'high' wie auf 'low' programmiert werden.

### Interruptkontrolle

In einigen Beispielen dieses Artikels wiesen wir schon auf die 3 Stufen des Interrupts hin. Den möglichen Interruptquellen des VIA (Steuerleitungen, Timer und Schieberegister) sind im Interrupt-Anzeigeregister IFR Flags zugeordnet. Sobald ein Flag auf '1' geht, wird dort auch Bit 7 logisch ODER auf '1' gesetzt. Das ermöglicht dem Prozessor mit dem BIT-Befehl eine schnelle Abfrage, ob ein Interrupt vom betreffenden Chip ausging. Wenn ja, dann kann in diesem IFR durch logische oder Verschiebebefehle die Interruptquelle im einzelnen schnell ausgemacht werden (Bild 9).

Eine hier angezeigte Interruptbedingung muß nicht notwendig zum Verlassen des im Vordergrund laufenden Maschinenprogramms führen. Welche Bedingungen – oder ob gar keine – Interrupt auslösen dürfen, wird durch Einschreiben in das IER, das Interrupt Enable Register festgelegt.

### Literatur

65xx MICRO MAG, Heft 7/79: Das VIA 6522. Rockwell International, Doc. No. 29650 N40: Versatile Interface Adapter (VIA).

### Stichworte zum Inhalt

VIA 6522, Timer, Interrupt, Schieberegister, Handshaking, Interface, I/O, E/A.



# Unsichtbares mischt mit

Wenn die Listings zweier Programme absolut deckungsgleich sind, dann sind auch die Programme identisch – so möchte man meinen. Daß diese Ansicht durchaus irrig sein kann, zeigte sich jüngst in Stuttgart, wo zwei Euro-Apple-Besitzer sich überraschend vor die Tatsache gestellt sahen, daß zwei Programme trotz gleicher Listings durchaus unterschiedliches Benehmen zeigten: Es ging um Steve Wozniaks „Animals“, ein „selbstlernendes“ Programm, das beim Erraten von Tiernamen von mal zu mal „dazulernt“ und sein neuerworbenes Wissen auf Diskette speichert. Identische Listings also; aber wo Programm Eins tat, was es sollte, da blieb bei Programm Zwei der Disketten-Antrieb stumm, und auf dem Schirm erschienen so befremdliche Texte wie „NOMON I, O, C“ oder auch mal „OPEN ANIMALSFILE, L 80“ – Dinge, die sicherlich ihren Sinn haben mögen, im „Animals-Spiel“ aber auf dem Schirm absolut nichts verloren haben. Was die Angelegenheit vollends mit einem Hauch von absurdem Theater versah: In beiden Programm-Versionen, der funktionierende und in der außer Rand und Band geratenen, hieß es in gewissen Zeilen ausdrücklich: PRINT „NOMON I, O, C“ und auch PRINT „OPEN ANIMALSFILE, L 80“.

Nächster Schritt bei der Suche nach der Lösung des Rätsels bildete die versuchsweise Entfernung der ominösen Zeilen aus der spielenden Programm-Version, nachdem die betreffenden Texte der Print-Anweisungen ja ohnehin nie auf dem Schirm auftauchte – und nun wurde alles noch viel merkwürdiger; jetzt blieb auch hier die Diskettenstation stumm und tot. Das nun führte zur Aufklärung.

Es stellte sich nämlich heraus, daß beide Texte Rechnerkommandos an das Disketten-Interface darstellen. Im „NOMON“-Fall wird dem Interface bedeutet, es habe den Daten- und Kommandofluß von und zur Diskette bitte schön nicht auf dem Bildschirm darzustellen, und beim „Open-Befehl“ ging's um die Eröffnung einer Datei. Damit

nun Disk-Operating-System und Controller wissen, daß sie gemeint sind, wenn ein derartiger PRINT-Befehl ergeht, und damit der Rechner selbst erfährt, daß er derartiges trotz PRINT nicht auf den Schirm zu printen hat, wird dem jeweiligen Text ein ASCII-Zeichen mit dem Dezimalcode 4 vorangestellt. Nach diesem Schlüssel-Byte wird solange ins Disk-Operating-System „geprintet“, bis ein Zeichen ‚Wagenrücklauf‘ die alten Verhältnisse wieder herstellt.

Das ASCII-Zeichen mit dem Dezimal-Abbild 4 (0000 0100) gehört zu den sogenannten nichtdruckenden Kontrollzeichen, die von den Rechner-Herstellern offenbar gelegentlich nach Belieben definiert werden. Beim PET bezeichnet es beispielsweise, hier mit dem Namen EOT (End of Tape) versehen, den Abschluß einer Bandaufzeichnung. Das Tückische an diesen „nichtdruckenden Zeichen“ nun, ist sie erscheinen weder in Drucker-Listings noch beim Programm-Auflisten auf dem Schirm. Sie sind sehr wohl gespeichert im Basic-Text, aber eben unsichtbar.

Wie macht man nun Unsichtbares sichtbar? Da gibt es einen Trick, der sofort zu klaren Verhältnissen verhilft. Hat man einen bestimmten Text-String in Verdacht, Kontrollzeichen neben sichtbarem Text zu enthalten, so drucke man den String aus, zähle die dargestellten Zeichen ab, und fordere den Rechner dann auf, den Wert der Funktion LEN (String) auszugeben. Ist diese Zahl größer als die der zuvor abgezählten Zeichen, dann ist der Beweis erbracht: Der String birgt Unsichtbares. In den meisten Fällen wird das Kontrollzeichen am Anfang stehen – und hier wird auch gleich gezeigt, wie man den Störenfried näher identifiziert; es hilft der Befehl PRINTASC (String) weiter – er liefert unmittelbar den dezimalen ASCII-Code für das erste Zeichen von (String). Übrigensw – „String“ steht hier und in den vorangegangenen Zeilen an Stelle des jeweils betrachteten Strings selbst, also für AS beispielsweise, oder „MUSTERSTRING“.

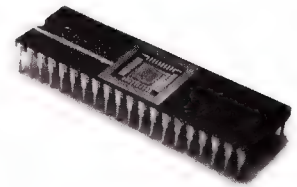
Es gibt druckende Zeichen und es gibt nichtdruckende ASCII-Zeichen. Ferner gibt es auf manchen Rechnern wie dem PET zum Beispiel, auch eine dritte Klasse von Zeichen, die auf dem Schirm zwar zur Darstellung von Symbolen führen, nicht jedoch auf angeschlossenen normgerechten ASCII-Druckern. Beim PET gehören die Symbole dazu, die das Schirmbild beeinflussen: Schirm löschen und Cursor-Bewegungen – wie zum Beispiel das ASCII-Zeichen mit dem Dezimal-Abbild 147. Wenn es außerhalb einer durch Anführungszeichen geschlossenen Kette gedrückt wird, löscht es den Schirm und schafft den Cursor in die Startposition – innerhalb von Anführungszeichen gesendet, wird das Zeichen durch ein schwarzes Herz-Symbol in weißem Feld dargestellt. Norm-Drucker nun, die nicht auf die PET-Eigenheiten vorbereitet sind, wissen mit diesem Bitmuster nichts anzufangen – sie lassen das Zeichen weg. Das bedeutet aber, daß Programm-Listing und Programm auch in diesem Fall nicht identisch sind.

Was ist gegen all das zu tun? Es gibt einen Ausweg, der vorzüglich geeignet ist, Unsichtbares gleich von Anfang an gar nicht ins Programm hineinzulassen. Uns hilft eine Funktion, über die heute nahezu alle moderneren Basic-Interpreter verfügen: die Anweisung CHR\$(N) nämlich. Sie stellt einen Ein-Zeichen-String zur Verfügung, der das Bitmuster mit dem ASCII-Dezimalcode N enthält. Setzt man in ein PET-Programm statt des Kommandos PRINT, gefolgt vom negativen Herzen in Anführungszeichen, die Anweisung PRINTCHR\$(147) ein, so ist der Effekt für den Programm-Ablauf der gleiche. Im Listing dagegen steht man nun vor der angenehmen Eigenschaft der betreffenden Zeile, daß sie mit jedem Wald- und Wiesen-Drucker lückenfrei darzustellen ist. Das gleiche gilt für den Euro-Apple und seine Kommandos ans Disketten-System: PRINTCHR\$(4) „NOMON I, O, C“ leistet das gleiche wie die Geisterzeile, die den Anlaß für dieser Beitrag brachte. –

Hans-Georg Joepgen



Fortschritte auf dem Gebiet der Mikroprozessoren zeigen sich in erster Linie am Befehlssatz dieser Intelligenz-Chips. Eine ideale Anpassung der Prozessorbefehle an die individuellen Bedürfnisse eines Anwenders ist aber kaum wirtschaftlich durchführbar. Der nachfolgende Beitrag zeigt eine Methode, wie ein handelsüblicher Mikroprozessor, der 6502, einen nicht existierenden „Idealprozessor“ simulieren kann.



# Der Individualisten-Prozessor

## Mikroprogramme

Viele Mikroprozessoren oder Mehrchip-Prozessoren führen die einzelnen Maschinenbefehle, also die binären bzw. hexadezimalen Operationscodes, nicht direkt per Hardware aus, sondern zerlegen jeden Befehl in mehrere Teilbefehle, die dann in Form eines „Mikroprogramms“ abgearbeitet werden.

Es ist ohne weiteres möglich, auch einen Mikroprozessor dazu zu bringen, eine Folge von Operationscodes nicht direkt auszuführen, sondern jeden Code als kleines Maschinenprogramm abzuarbeiten. Die Operationscodes sind dann nicht seine eigenen Maschinenbefehle, sondern werden von einem besonderen Steuerprogramm nur als Folge solcher interpretiert. Das Steuerprogramm kann man daher auch als „Interpreter“ bezeichnen – es arbeitet nach dem gleichen Prinzip wie ein üblicher Basic-Interpreter.

## Der Interpreter

Bild 1 zeigt das Interpreter-Programm. Es holt sich die einzelnen Operationscodes (nicht die des 6502, sondern die vom Anwender selbst definierten) vom Programmspeicher, der hier bei 0200 beginnt. In einer Tabelle – sie beginnt hier bei 0000 – wird jedem Operationscode eine Adresse zugewiesen, an der das jeweilige Programm zum Abarbeiten des Codes steht. Dank eines kleinen Programmierkniffes benötigen diese Programme keinen 3-Byte-Rücksprungbefehl, sondern können mit RTS (hex 60) abgeschlossen werden.

Das Interpreter-Programm läßt maximal 128 unterschiedliche Operationscodes zu, nämlich 00...7F. Setzt man das höchstwertige Bit der Codes auf 1 (80...FF), so wird eine Debugging-Routine angesprungen, die z. B. dazu dienen kann, bei jedem Befehl kurz die momentane Adresse des Anwenderprogramms und den dort stehenden Operationscode auf dem Display des Mikrocomputers KIM-1 anzuzeigen (Bild 2). Drückt man irgendeine Taste auf dem KIM, so kann man den Ablauf auch ganz anhalten, bis die Taste wieder losgelassen wird. Von sol-

chen Debugging-Möglichkeiten kann ein realer Mikroprozessor nur träumen! Eine weitere Möglichkeit wäre z. B. eine Art „Trace“-Betriebsart, bei der Adresse und Operationscode auf einem Terminal ausgedruckt werden. Das Anwenderprogramm selbst wird – außer in seiner Geschwindigkeit – von der Debugging-Betriebsart nicht beeinflusst. Da der Anwender gezielt bei ein-

zelnen Befehlen des Programms das höchstwertige Bit auf 1 setzen kann, ist ein sehr komfortabler Trace-Betrieb in Teilbereichen des Programms möglich.

## Die Adressentabelle

Wie schon erwähnt, beginnt bei der Adresse 0000 eine Tabelle, die die Unterprogrammadressen zur Abarbeitung

00A0	A2	FF	LDX	FF	S T A R T
00A2	9A		TXS		STACKPNT=FF
00A3	E8		INX		PROGRAMM-
00A4	86	EF	STX	EF	ZAEHLER
00A6	A2	02	LDX	02	=0200
00A8	86	F0	STX	F0	
00AA	08		PHP		STATUS,
00AB	48		PHA		X, A RETTEN
00AC	8A		TXA		
00AD	48		PHA		
00AE	A2	00	LDX	00	OP-CODE
00B0	A1	EF	LDA	(EF, X)	HOLEN
00B2	0A		ASL	A	CODE MAL 2
00B3	AA		TAX		ALS TAB-INDEX
00B4	90	03	BCC	00B9	CODE >7F:
00B6	20	80	JSR	0080	DEBUG-PGM
00B9	BD	00	LDA	0000, X	ADL UND
00BC	85	F4	STA	F4	ADH UM-
00BE	E8		INX		SPEICHERN
00BF	BD	00	LDA	0000, X	
00C2	85	F5	STA	F5	
00C4	E6	EF	INC	EF	PROGRAMM-
00C6	D0	02	BNE	00CA	ZAEHLER
00C8	E6	F0	INC	F0	ERHOEHEN
00CA	68		PLA		
00CB	AA		TAX		ALLES
00CC	68		PLA		RUECK-
00CD	28		PLP		SPEICHERN
00CE	20	D4	JSR	00D4	JSR INDIR.
00D1	4C	AA	JMP	00AA	
00D4	6C	F4	JMP	(00F4)	

Bild 1. Das Interpreter-Steuerprogramm interpretiert die Operations-Codes des „virtuellen“ Prozessors als Unterprogramme, deren Adressen in einer ab 0000 stehenden Tabelle stehen. Operationscodes, deren höchstwertiges Bit 1 ist, führen zu einem „Umweg“ über die Debugging-Routine an der Adresse 0080



der Operationscodes enthält, und zwar in der Reihenfolge ADL, ADH. Der Tabellenplatz ergibt sich durch Multiplizieren des Operationscodes mit 2; so steht etwa die Adresse des Unterprogramms für den Operationscode 04 an der Tabellenadresse 0008. Die Adressentabelle kann maximal 256 Byte, also eine „Page“ umfassen, so daß die Operationscodes 00...7F möglich sind (das höchstwertige Bit dient ja der Trace-Betriebsart, s. o.). Bei der hier gewählten Speicherbelegung ist der Tabellenraum natürlich eingeschränkt, da in der „Zero Page“ für das Interpreterprogramm und Daten schon einige Adressen belegt sind. In einem 6502-System mit erweitertem Speicher kann die Tabelle aber woanders hingelegt werden, ebenso braucht der Interpreter nicht in der Zero-Page stehen.

Ein Beispiel zur Adressentabelle: Will man dafür sorgen, daß beim Operationscode 04 ein Unterprogramm an der Adresse 1E5A angesprungen wird (beim KIM-1 kann man damit ein ASCII-Zeichen vom Terminal holen), so muß man in die Tabelle bei 0008 die Daten 5A und bei 0009 1E hineinschreiben.

### Der Programmzähler

Wie jeder Mikroprozessor, so braucht auch unser „virtueller“, simulierter Prozessor einen Programmzähler. Er wird von zwei aufeinanderfolgenden Zero-Page-Adressen dargestellt, nämlich 00EF und 00F0. Der Interpreter hat den Programmzähler bereits um 1 inkrementiert, wenn das Unterprogramm zur Abarbeitung des Operationscodes angesprungen wird. Bei Mehr-Byte-Befehlen kann so das dem Operationscode folgende Byte z. B. als Datenbyte ausgewertet werden. Das jeweilige Unterprogramm muß aber selbst dafür sorgen, daß – wenn der decodierte Befehl mehr als 1 Byte umfaßt – der Programmzähler zusätzlich inkrementiert wird. Bei 1-Byte-Befehlen braucht man sich um den Programmzähler nicht weiter zu kümmern.

0000 4F 1C	00=SPRUNG ZUM KIM
0002 A0 00	01=RESTART BEI 0200
0004 2F 1E	02=CRLF AUSGEBEN
0006 70 00	03=SPACE AUSGEBEN
0008 5A 1E	04=ASCII-CHR. HOLEN
000A 3B 1E	05=BYTE AUSGEBEN
0070 48	PHA SPACE
0071 20 9E 1E	JSR 1E9E AUSGEBEN,
0074 68	PLA AKKU
0075 60	RTS RETTEN
0200 82 84 83 85 81	(DEBUG)
0200 02 04 03 05 01	(NORMAL)

Bild 2. Debugging-Routine für den KIM-1. Bei jedem Befehl wird während des Programmablaufes auf dem KIM-Display kurz die Adresse und der dort stehende Operationscode dargestellt

0080 98	TYA	DEBUG-
0081 48	PHA	ROUTINE
0082 8A	TXA	
0083 48	PHA	X UND Y
0084 A2 Fo	LDX Fo	WERDEN
0086 8A	TXA	GERETTET
0087 48	PHA	
0088 A5 EF	LDA EF	ANZEIGE
008A 85 FA	STA FA	VON ADR.
008C A5 Fo	LDA Fo	UND OP.-
008E 85 FB	STA FB	CODE
0090 20 19 1F	JSR 1F19	
0093 D0 FB	BNE 0090	HALT BEI
0095 68	PLA	GEDR. KIM-
0096 AA	TAX	TASTE
0097 CA	DEX	
0098 D0 EC	BNE 0086	VERZOEG.-
009A 68	PLA	SCHLEIFE
009B AA	TAX	(ZEIT:
009C 68	PLA	0085)
009D A8	TAY	
009E 60	RTS	

### Stack-Verwendung

Um Platz in der Zero-Page und im RAM ab 0200 zu sparen, wurde ausgiebiger Gebrauch von der Möglichkeit gemacht, Zwischenwerte und Registerinhalte auf dem Stack zu retten. Das Interpreter-Programm selbst verändert weder den Akku noch das X-Register, das Y-Register und den Status. Dadurch ist es möglich, daß ein Operationscode dem nächsten Variablenwerte in einem der Register übergibt. Auch bedingte Sprungbefehle lassen sich leicht implementieren, da ja das Statusregister zwischen den einzelnen Operationscodes des Anwenderprogramms und damit zwischen den Interpreter-Unterprogrammen nicht verändert wird.

### Jetzt wird's individuell

Außer dem Interpreter-Steuerprogramm können die Operationscodes des virtuellen Prozessors, die Unter-

programme für ihre Abarbeitung und damit auch die Unterprogramm-Adressentabelle individuell optimal an einen bestimmten Verwendungszweck angepaßt werden. Trotzdem sind einige Überlegungen hinsichtlich einer günstigen Verwendung der Operationscodes angebracht.

Zunächst einmal muß man sich über Sinn und Grenzen eines solchen Mikroprogramm-Interpreters klar werden. Die erreichbare Geschwindigkeit liegt deutlich unter der eines „nackten“ Prozessors, da ja auch das Interpreter-Steuerprogramm Zeit benötigt. Hinsichtlich der Geschwindigkeit kann der virtuelle Prozessor daher umso eher mit dem Mutterprozessor konkurrieren, je länger die Unterprogramme für die einzelnen Operationscodes sind, je komfortabler also die Befehle der simulierten CPU sind. Dann ist nämlich der Zeitbedarf des Steuerprogramms im Vergleich zu dem der Unterprogramme relativ gering.

Es ist aber ohnehin nicht der Zweck des Interpreters, eine möglichst hohe Verarbeitungsgeschwindigkeit zu erzielen. Vielmehr soll die Programmierung bei Problemstellungen, die nicht allzu kritisch in bezug auf die Ausführungszeit der Befehle sind, durch möglichst komfortable Befehle vereinfacht werden. Wie das in der Praxis aussieht, sei hier zum Abschluß an einem einfachen Beispiel gezeigt. Es ist für den Betrieb des Mikrocomputers KIM-1 an einem ASCII-Terminal ausgelegt, eignet sich nach geringfügigen Änderungen aber auch z. B. für den SYM-1 oder AIM-65 bzw. PC-100.

Bild 3. Ein einfaches Anwendungsbeispiel. Mit nur wenigen 1-Byte-Befehlen läßt sich ein ASCII-Zeichen in sein Hex-Äquivalent umrechnen. Die Adressen der Monitor-Unterprogramme beziehen sich wieder auf den KIM-1



## Ein Beispiel

Wie sehr ein Interpreter die Programmierung vereinfachen kann, wird schnell an unserem Beispiel deutlich. Bild 3 zeigt eine einfache Adressentabelle und ein dabei verwendetes Unterprogramm. Alle anderen Unterprogramme stehen im ROM des KIM-Monitors.

Das Beispielprogramm an der Adresse 0200 gibt auf dem Terminal einen Carriage Return/Line Feed aus, holt dann von der Terminal-Tastatur ein

jetzt einzutippendes ASCII-Zeichen (z. B. A), läßt dann einen Leerraum folgen und druckt schließlich das dem ASCII-Zeichen entsprechende Hex-Äquivalent aus (z. B. 41). Danach erfolgt ein Rücksprung zum Programm-anfang, und das Spiel beginnt von neuem. Das Programm umfaßt nur 5 Byte.

In unserem Beispiel wurden ausschließlich 1-Byte-Befehle verwendet. Schon daraus wird ersichtlich, wie komfortabel die Befehle gestaltet wer-

den können. (Wie Mehr-Byte-Befehle realisiert werden, ist weiter oben beschrieben.)

Der Individualität sind jedenfalls keine Grenzen gesetzt. Versuchen Sie mal, Ihren eigenen 32-bit-Prozessor zu „bauen“!

Herwig Feichtinger

## Stichworte zum Inhalt

6502, KIM-1, Interpreter, virtuelle CPU, Mikroprogramme, Operationscodes, Befehls-satz.

# SYM druckt 16 Byte pro Zeile

Der Mikrocomputer SYM-1 bietet die Möglichkeit, mittels des Verify-Befehls einen vorwählbaren Speicherbereich ausdrucken zu lassen bzw. auf einem Terminal sichtbar zu machen. Pro Zeile werden hierbei 8 Byte ausgegeben, was hauptsächlich bei Verwendung eines Monitors nachteilig ist, da die gesamte rechte Hälfte des Bildschirms ungenutzt bleibt.

Das folgende Programm listet einen vorwählbaren Speicherbereich in Zeilen zu je 16 Byte auf. Daraus ergibt sich eine verbesserte Nutzung des Bildschirms. Auf die Ausgabe der Prüfsumme wurde in diesem Programm verzichtet.

Nach Eintasten des Programms wird es mit „GO 300“ oder „JUMP 5“ gestartet. Es wird folgender Text ausgelesen:

## PROGRAMMAUFLISTUNG

BITTE GEBEN SIE ANFANGS-ADRESSE UND ENDADRESSE + 1 EIN. Diese beiden Adressen werden durch ein Minuszeichen getrennt eingegeben. Nach Betätigen der Taste CR listet das Programm den gewählten Bereich in Zeilen zu je 16 Byte auf, wobei am Zeilenanfang die jeweilige Adresse des folgenden Bytes angegeben wird. Die Anzahl der Bytes pro Zeile ist in Adresse 033D festgelegt.

Die Auslesegeschwindigkeit ist wählbar zwischen 110 Baud und 4800 Baud. Sie wird vom Inhalt der Adresse 030E bestimmt und ist in diesem Programm für 300 Baud ausgelegt. Für andere Geschwindigkeiten wird das Byte geändert: D5 für 110 Baud, 24 für 600 Baud, 10 für 1200 Baud, 06 für 2400

Baud und 01 für 4800 Baud. Wenn nicht soviel Speicherplatz zur Verfügung steht und auf den Einleitungstext (0200–02FF) verzichtet werden soll, ist folgendermaßen vorzugehen: 0300–031C bleiben unbenutzt; 031D wird abgeändert von 4C 14 03 in 20 86 8B.

In diesem Fall wird das Programm mit „GO 031D“ gestartet. Nach Eingabe der Anfangsadresse und Endadresse + 1 – durch Minuszeichen getrennt – listet das Programm nach Betätigen von „CR“ den gewählten Speicherbereich wie vor auf. In diesem Fall muß vor Programmstart die Vektoränderung SD 8AA0, A664 vorgenommen und in SDBYT (A651) das entsprechende Geschwindigkeitsbyte eingegeben werden.

Hans Eckert

0300 20 86 8B JSR 8B86	032A 20 47 8A JSR 8A47	0357 EE 4D A6 INC A64D
0303 A9 A0 LDA =A0	032D AD 4D A6 LDA A64D	035A AD 4C A6 LDA A64C
0305 8D 64 A6 STA A664	0330 20 FA 82 JSR 82FA	035D CD 4A A6 CMP A64A
0308 A9 8A LDA =8A	0333 AD 4C A6 LDA A64C	0360 Fo 13 BEQ 0375
030A 8D 65 A6 STA A665	0336 20 FA 82 JSR 82FA	0362 CA DEX
030D A9 4C LDA =4C	0339 20 42 83 JSR 8342	0363 Fo 03 BEQ 0368
030F 8D 51 A6 STA A651	033C A2 10 LDX =10	0365 4C 40 03 JMP 0340
0312 A0 00 LDY =00	033E A0 00 LDY =00	0368 A9 0D LDA =0D
0314 B9 00 02 LDA 0200, Y	0340 AD 4D A6 LDA A64D	036A 20 47 8A JSR 8A47
0317 Fo 07 BEQ 0320	0343 85 01 STA 01	036D A9 0A LDA =0A
0319 20 47 8A JSR 8A47	0345 AD 4C A6 LDA A64C	036F 20 47 8A JSR 8A47
031C C8 INY	0348 85 00 STA 00	0372 4C 2D 03 JMP 032D
031D 4C 14 03 JMP 0314	034A B1 00 LDA (00), Y	0375 AD 4D A6 LDA A64D
0320 20 20 82 JSR 8220	034C 20 FA 82 JSR 82FA	0378 CD 4B A6 CMP A64B
0323 A9 0D LDA =0D	034F 20 42 83 JSR 8342	037B Fo 03 BEQ 0380
0325 20 47 8A JSR 8A47	0352 EE 4C A6 INC A64C	037D 4C 62 03 JMP 0362
0328 A9 0A LDA =0A	0355 D0 03 BNE 035A	0380 20 72 89 JSR 8972
		0383 4C 00 80 JMP 8000

Programmauflistung (0300...0385) und Textblock (0200...0254) des Dump-Programms für den SYM-1. Die Startadresse ist 0300

0200 0D 0A 50 52 4F 47 52 41 4D 4D 41 55 46 4C 49 53
0210 54 55 4E 47 20 20 20 20 20 20 20 20 20 0D 0A
0220 42 49 54 54 45 20 47 45 42 45 4E 20 53 49 45 20
0230 41 4E 46 41 4E 47 53 41 44 52 45 53 53 45 20 55
0240 4E 44 20 45 4E 44 41 44 52 45 53 53 45 2B 31 20
0250 45 49 4E 2A 00



# Amerika–

## so weit wie der nächste Briefkasten

Der Markt für Hobbycomputer-Zubehör hat sich in den letzten Monaten in der Bundesrepublik stark ausgeweitet – und dies nicht nur, was das Volumen der getätigten Abschlüsse angeht; auch die Zahl der angebotenen Produkte aus dem Bereich Hardware und Software nahm hierzulande stark zu. Dennoch – verglichen mit dem, was sich zur Zeit in den Vereinigten Staaten tut, muß das Angebotsspektrum in Europa dagegen noch immer als ausgesprochen dürftig gelten.

Ein Teil der amerikanischen „Renner“ – Rechner, Zubehör, Software – erschien unterdessen in den Angebotslisten bundesdeutscher Firmen. Allerdings, wenn sich – beispielsweise nach der Lektüre von Anzeigen – ein Käufer zum Bezug der angebotenen Ware entschließt, dann muß er – wie es dem Autor im ersten Halbjahr 79 wiederholt passierte – oft genug zweierlei registrieren: Der Anbieter hat die Ware, für die er Käufer sucht, noch längst nicht im Hause und vertröstet auf Liefertermine, die noch wochenweit in der Zukunft liegen und auf deren Einhaltung die Firma sich nicht vertraglich festlegen lassen will, oder – zum zweiten – die bundesdeutschen Preise erscheinen, verglichen mit den aus amerikanischen Zeitschriften bekannten Original-USA-Preisen, auch dann noch als saftig überhöht, wenn man berücksichtigt, daß der Importeur neben seiner Verdienstspanne auch noch Unkosten für Versand und Verzollung in den Preis einkalkulieren muß.

Einen Ausweg aus dieser unguten Situation bietet der „Eigenimport“ – der Direktbezug von Hardware und Software aus den USA. Das ist eine Form des Einkaufs, die man auf den ersten Blick für umständlich und schwierig halten muß, denn schließlich gehören die Vereinigten Staaten nicht zum EG-Wirtschaftsraum, und dies erschwert Direkteinkäufe zumindest für Kunden ohne Erfahrungen mit „transkontinentalen Geschäftstransaktionen“.

Dennoch sind Direktkäufe trotz allem weniger schwierig abzuwickeln, als es zuerst den Anschein hat, besonders in jenen Fällen, in denen es sich beim Anbieter um ein als vertrauenerweckend eingeschätztes Unternehmen mit gu-

tem Namen handelt, mit dem man auf der Geschäftsbeziehungsbasis „Vorkasse“ verkehren kann. Da hat sich beim Autor folgendes Verfahren bewährt, das allerdings nur Inhabern eines Postscheckkontos zugänglich ist. Dieser Einschränkung kommt allerdings nur geringes Gewicht zu, da jedermann gegen mäßige Gebühr innerhalb weniger Tage ein solches Konto durch Rücksprache auf jedem beliebigen Postamt einrichten kann und ein solches Konto, verglichen mit üblichen Bank-Girokonten, ohnehin mancherlei Vorteile bringt.

Man füllt ein gewöhnliches Postscheck-Überweisungsformular mit der Adresse des Zahlungsempfängers in den Vereinigten Staaten aus und trägt in den DM-Spalten des Formulars den geforderten Dollar-Betrag ein, wobei man die Währungsbezeichnung DM durchstreicht und durch „US-Dollar“ ersetzt. Hierbei empfiehlt es sich, um zwei, drei Dollar nach oben aufzurunden, damit die Unkosten des Verkäufers für den üblich weiten Versandweg gedeckt werden (bei schwereren Waren entsprechend mehr). Auf dem Abschnitt für den Zahlungsempfänger vermerkt man das gewünschte Produkt und gegebenenfalls noch den Versandweg (zum Beispiel „Air Mail Please“, wenn es um Bücher oder andere leichte Gegenstände wie Tonbandkassetten geht). Den Rest erledigt die Deutsche Bundespost, und zwar unabhängig davon, ob der Zahlungsempfänger Postscheckteilnehmer ist oder nicht. Es wird nach dem Tageskurs die DM-Entsprechung der Dollarsumme dem Postscheckkonto des Bestellers belastet, und die Post veranlaßt über eine Korrespondenzbank in den Staaten, daß der Verkäufer sein Geld bekommt. Der schickt dann die Ware auf dem Postweg los, und die Post zieht beim Besteller dann auch die Zollgebühren ein. Wie eine Wirtschaftlichkeitsberechnung in einem solchen Fall aussieht und welche Gebühren dabei entstehen, wird am folgenden Beispiel erläutert.

Ein bestimmtes Druckerinterface, für das ein deutscher Händler knapp 700 Mark haben will, wird im März-Heft einer amerikanischen Hobby-Computer-Zeitschrift für 59,75 Dollar angeboten. Im April überweist der Autor 62

Dollar an den Anbieter. Der Post bucht dafür, einschließlich Gebühren in Höhe von 3,20 DM, genau 120,07 DM vom Postscheckkonto des Empfängers ab. Einige Wochen darauf kommt ein Paket: Gegen Zahlung von 26,70 DM (15 Prozent Zoll für „Computerteile“, berechnet aus dem neuen Dollar-Tageskurs, sowie eine neue Postgebühr für die Abwicklung der Zollformalitäten), wird das Druckerinterface ausgehändigt.

Die Ersparnis im vorliegenden Fall ist eklatant; hier wurde für ein Produkt, das ein deutscher Händler für rund 680 Mark anbot, nur ein Betrag von zusammen knapp 147 Mark bezahlt.

Zu diesem Beispiel ist nun zweierlei zu bemerken: Einmal ist die Preisdifferenz zwischen dem amerikanischen und dem deutschen Binnenangebot nicht in jedem Fall so ärgerlich hoch wie beim Druckerinterface, und dann sind mit einer solchen Form des Einkaufs zusätzliche Nachteile verbunden. Der Zahlungsweg Vorkasse kann zum Verlust der Kaufsumme führen, wenn man an einen unseriösen Verkäufer gerät, denn das Einklagen des gezahlten Betrages über den Atlantik hinweg stößt auf außerordentliche Schwierigkeiten und macht hohe Kosten. Zum zweiten kann es Probleme geben, wenn die Ware Mängel aufweist; hier sein gutes Recht zu bekommen, ist in aller Regel schwieriger als nach einem Kauf beim einheimischen Händler.

Man wird also in jedem Fall abwägen müssen, bevor man sich zum Eigenimport entschließt. Was den Autor angeht, so hat er im ersten Halbjahr 79 in neun Fällen Waren aus Amerika bezogen: Hardware, Software, Bücher. Nur in einem einzigen Fall ließ ein Verkäufer nach dem Einstreichen der Kaufsumme drei Monate lang nichts mehr von sich hören, und nur in einem einzigen weiteren Fall entstanden Zusatzkosten durch Austausch eines Transformators und eines Netzsteckers, weil aus den USA bezogene Hardware natürlich auf dortige Netzverhältnisse eingerichtet ist. Dennoch: Die Schlußrechnung bei den Eigenimporten des Verfassers weist einige hundert Mark an Ersparnis aus.

Hans-Georg Joepgen



Gerd Duddek



Zu einer einfachen Lösung für ein Baudot-Interface kommt man, wenn man die „Denkarbeit“ den Mikroprozessor des TRS-80 selbst ausführen läßt. Daß dieses möglich ist, zeigt ein Blick in das Manual zu der von TANDY angebotenen RS-232-Schnittstelle für das TRS-80-System. Die recht ausführliche Softwarebeschreibung zeigt auf, wie der BASIC-Interpreter des TRS-80 (Level II) einen Drucker bedient. Die Adresse des zur Ansteuerung eines Druckers benötigten Unterprogrammes (sog. Treiber) ist in Form eines „Device Control Block“ (DCB) im RAM abgelegt, beginnend bei der Adresse 16421 (4025H) (siehe auch Anhang D im Handbuch für BASIC Level II). Beim Einschalten des Mikrocomputers wird hier die Adresse des für den Original-Drucker gedachten Treibers, der im BASIC-ROM enthalten ist, abgelegt. Wird der DCB nachträglich geändert, so kann man dadurch ein eigenes Unterprogramm adressieren, das im RAM abgelegt ist und in der Lage ist, eine Baudot-Fernschreibmaschine zu treiben.

Dieses Maschinenprogramm muß natürlich in einem RAM-Bereich liegen, der nicht vom BASIC-Interpreter überschrieben werden kann. Eine derartige Abgrenzung eines RAM-Bereiches ist beim TRS-80-System durch eine entsprechende Antwort auf die Frage „MEMORY SIZE?“ möglich, die das System nach dem Einschalten stellt.

Die folgende Hard- und Softwarebeschreibung zeigt nun eine Möglichkeit zur Realisierung der angestellten Überlegungen auf.

## Hardware

Bild 1 zeigt das Schaltbild für ein einfaches Baudot-Interface, das speziell auf den Expansion-Bus des TRS-80-Systems zugeschnitten ist. Das Interface kann von dem System unter der Adresse 03 angesprochen werden. Während der Ausführung eines derartigen Input- oder Outputbefehles setzt die CPU eine 8-bit-Adresse für den anzusprechenden Port auf die untere Hälfte des Adressen-Bus, wobei gleichzeitig die IN- bzw. OUT-Leitung des TRS-80-Bus L-Pegel führt. Dem entsprechend bildet das 8-Eingang-NAND-Gatter 74LS30 zusammen mit den 6 Invertern des 74LS04 eine Decodierung für die Adresse 03. Das mittels eines NOR-Gatters mit dem OUT-Signal verknüpfte Ausgangssignal des 74LS30 steuert die Gate-Eingänge von zwei 4-bit-Latches (74LS75). Diese bilden so einen 8-bit-Outputport. Das mit dem IN-Signal NOR-verknüpfte Ausgangssignal des 74LS30 steuert die Enable-Eingänge eines 4fach Bus-Puffers 74LS126, der somit einen 4-bit-Input-Port bildet. Die in Bild 1 für die Anschlüsse verwendeten Bezeichnungen und Numerierungen entsprechen der

Pin-Belegung des TRS-80 Expansion-Port (siehe Anhang im Level-I Handbuch).

Zur Erzeugung von seriellen Fernschreibzeichen werden die 5 niederen Bits des Outputports an einen Parallel/Serienwandler weitergegeben. Die restlichen 3 Bits (F0...F2), sowie 3 Bits des Inputports (E0...E2), werden im Moment nicht benötigt, stehen aber – bei Vorhandensein der notwendigen Software – für andere Zwecke zur Verfügung.

```

10 REM BAUDOT-TEST
20 REM VON G.DUDEK
30 REM TEST DES BAUDOT
  -INTERFACE
35 A=31
37 GOSUB200
40 A=8
50 GOSUB200
60 A=2
70 GOSUB200
80 FOR I=1 TO 32
90 A=10
100 GOSUB200
110 A=21
120 GOSUB200
130 NEXT
140 GOTO 35
150 STOP
200 REM BEDIENUNG DES
  BAUDOT-INTERFACE
210 REM FRAGE, OB INTERFACE
  BEREIT
220 S=INP(3)
230 IF (S AND 1)=1 THEN 220
240 REM ZEICHEN AN
  INTERFACE AUSGEBEN
250 OUT 3,A
260 RETURN

```

Bild 3. Testprogramm für das Interface

Als Parallel/Serienwandler dient das Datenselektor-IC 74LS151. Dieses IC schaltet jeweils denjenigen seiner D-Eingänge auf den Y-Ausgang durch, dessen „Hausnummer“ in binärer Form an den Datenselektoreingängen A, B, C liegt. Der mittels eines aus zwei NOR-

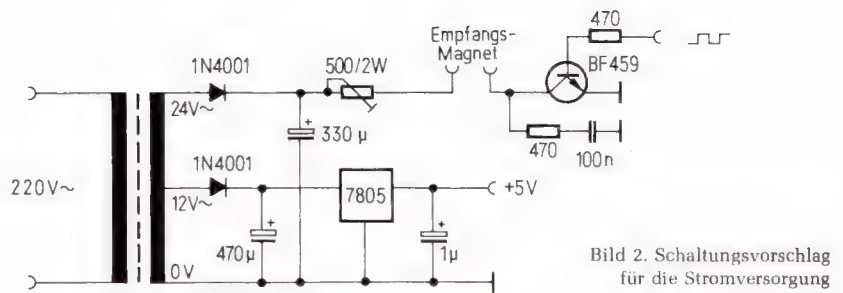


Bild 2. Schaltungsvorschlag für die Stromversorgung

Gattern gebildeten RS-Flipflops als 7-Zähler geschaltete 74LS90 sorgt dafür, daß nacheinander alle D-Eingänge an die Reihe kommen. Dabei bilden die Eingänge D0 und D7 die Stoppstritte und D1 den Startschritt für die Fernschreibmaschine. Den Takt bezieht der Zähler aus einem Timer-IC 555, das als astabiler Multivibrator geschaltet ist. Seine Frequenz in Hz muß gleich der Baud-Rate der verwendeten Fernschreibmaschine sein. Die Frequenz läßt sich mit dem 50-kΩ-Trimpotentiometer einstellen, das deshalb als Spindeltrimmer ausgeführt werden sollte. Da der 0,68-µF-Kondensator ebenfalls frequenzbestimmend ist, sollte hier ein Folienkondensator verwendet werden.

Das RS-Flipflop ist so geschaltet, daß es beim Ausgeben eines Fernschreibzeichens an den Outputport zurückgesetzt wird und dadurch den Zähler freigibt. Gleichzeitig wird das niedrigste Bit des Inputports auf H-Pegel gesetzt. Dies signalisiert, daß das Interface mit der Ausgabe eines Zeichens beschäftigt ist und im Moment kein neues Zeichen annehmen kann. Sobald der Zähler auf den Stand 8 springt, wird das RS-Flipflop wieder gesetzt, was ein Rücksetzen und Blockieren des 74LS90 bewirkt, solange, bis ein neues Zeichen ausgegeben wird.

Wie bereits angedeutet, stehen die Fernschreibzeichen am Y-Ausgang des 74LS151 zur Verfügung. Sie können von hier aus auf eine bereits vorhandene Taststufe für den Linienstrom gegeben werden. Für den Fall, daß eine Linienstromversorgung noch nicht zur Verfügung steht, zeigt Bild 2 einen einfachen Schaltungsvorschlag für eine komplette Stromversorgung einschließlich einer Taststufe für den Linienstrom. Als Netztransformator verwendet man vorzugsweise einen zur Printplattenmontage geeigneten Typ, der eine Sekundärspannung von 2 x 12 V und ca. 300 mA liefert. Das Baudot-Interface kann dann einschließlich Netzteil auf einer Europakarte untergebracht werden. Für den Aufbau solcher Schaltungen haben sich beim Verfasser mit Kupferbahnen versehene Lochrasterplatten (2,54-mm-Raster) bewährt. Querverbindungen werden dabei in Fädelschleife ausgeführt.



Wird das Netzteil mit auf der Platine untergebracht, so ist besonders auf ausreichende Sicherheitsabstände bei „Hochspannung“ führenden Leitungen (Netzleitung, Linienstromversorgung) zu achten. Falls eine externe Linienstromquelle benutzt wird, sollte die Tastung des Linienstromes aus Sicherheitsgründen über ein Tastrelais erfolgen.

Bevor das Interface an den TRS-80 angeschlossen wird, sollte zunächst mit dem 500-Ω-Trimpotentiometer der richtige Linienstrom (meist 40 mA) eingestellt werden. Steht ein Frequenzmeßgerät zur Verfügung, so kann der Taktgenerator auf die richtige Frequenz gebracht werden. Nach Anschluß an den Mikrocomputer kann das Interface mit dem in Bild 3 aufgelisteten BASIC-Programm getestet werden. Bei richtiger Funktion muß die Fernschreibmaschine einwandfreie Zeilen mit „ryry...“ schreiben. Treten Fehldrucke auf, so kann das daran liegen, daß die Taktfrequenz des Parallel/Serienwandlers nicht richtig an die Schreibgeschwindigkeit der Fernschreibmaschine angepaßt ist. Möglicherweise befindet sich auch der Empfänger der Maschine in einer ungünstigen Stellung.

## Software

Um die Fernschreibmaschine auf die Lineprinter-Befehle des BASIC-Interpreters ansprechen zu lassen, fehlt jetzt nur noch ein geeignetes Treiberprogramm. Bild 4 zeigt das Assembler-Listing eines derartigen Programmes, das mit dem von TANDY gelieferten TRS-80 Editor/Assembler erstellt wurde. Dieses Listing wurde, wie die anderen hier vorgestellten Listings auch, auf einer alten Lorenz-Lo15-Fernschreibmaschine ausgedruckt. Da jedoch der Editor/Assembler ein eigenes Drucker-Treiberprogramm verwendet, wurde hier ein externes Mikroprozessorsystem benutzt, um die Fernschreibmaschine zu treiben.

Das Listing des Treiberprogrammes ist ohne Besonderheiten, jedoch dürften einige kurze Hinweise es erleichtern, seine Arbeitsweise zu verstehen. Das zu druckende ASCII-Zeichen befindet sich im C-Register der CPU. Zunächst wird überprüft, ob es sich um ein „Carriage Return“ (CR) handelt. Ist dies der Fall, so wird in ein als Hilfszeichen deklariertes Byte der Baudot-Code für „Wagenrücklauf“ geladen, in das als Zeichen benannte Byte der Baudot-Code für „Zeilenvorschub“. Anschließend werden diese beiden Bytes an das Baudot-Interface ausgegeben. Handelte es sich nicht um „CR“, so wird das ASCII-Zeichen zur relativen Adressierung einer Tabelle benutzt, in der die zugehörigen Baudot-Codes abgelegt sind.

7f30 f5	7f89 dd77ff	7fc7 0a
7f31 dde5	7f8c db03	7fc8 05
7f33 d5		7fc9 10
7f34 dd21b57f	7f8e e601	7fca 07
7f38 dd56fc	7f90 20fa	7fcb 1e
7f3b 79		7fcc 13
	7f92 dd7efd	7fcd 1d
7f3c fe0d	7f95 fe00	7fce 15
7f3e 200e	7f97 2808	7fcf 11
7f40 3ee8	7f99 d303	7fd0 99
7f42 dd77fd		7fd1 99
7f45 3ee2	7f9b db03	7fd2 99
7f47 dd77fe	7f9d e601	7fd3 99
7f4a 1642	7f9f 20fa	7fd4 99
7f4c 183e		7fd5 04
7f4e 79	7fa1 dd7efe	7fd6 9c
7f4f e63f	7fa4 d303	7fd7 85
7f51 32567f	7fa6 15	7fd8 99
7f54 dd7e00	7fa7 2897	7fd9 99
	7fa9 dd72fc	7fda 99
7f57 dd77fe		7fdb 99
7f5a e680	7fac d1	7fdc 85
7f5c 2018	7fad dde1	7fdd 8f
	7faf f1	7fde 92
7f5e dd7eff	7fb0 c9	7fdf 99
7f61 fe00		7fe0 91
7f63 2005	7fb5 1f	7fe1 8c
7f65 dd77fd	7fb6 03	7fe2 83
7f68 1822	7fb7 19	7fe3 9c
7f6a 3eff	7fb8 0e	7fe4 9d
7f6c dd77fd	7fb9 09	7fe5 96
7f6f 3e00	7fba 01	7fe6 97
7f71 dd77ff	7fbb 0d	7fe7 93
	7fbc 1a	7fe8 81
7f74 1816	7fbd 14	7fe9 8a
7f76 dd7eff	7fbe 06	7fea 90
7f79 fe00	7fbf 0b	7feb 95
7f7b 2807	7fc0 0f	7fec 87
	7fc1 12	7fed 86
7f7d 3e00	7fc2 1c	7fee 98
7f7f dd77fd	7fc3 0c	7fef 8e
7f82 1808	7fc4 18	7ff0 9c
7f84 3efb	7fc5 16	7ff1 8f
7f86 dd77fd	7fc6 17	7ff2 9e
		7ff3 92
		7ff4 99

Bild 4. Assembler-Listing des Drucker-Treiberprogrammes

Gibt es zu einem ASCII-Zeichen kein passendes Baudot-Zeichen, so wird durchweg ein Fragezeichen verwendet. Eine Ausnahme bilden die spitzen Klammern – „<“ und „>“ –, an deren Stelle gleichgerichtete runde Klammern – „(“ und „)“ – verwendet wer-

den. Durch dieses Verfahren bleiben Listings von BASIC-Programmen trotz des eingeschränkten Zeichensatzes noch einigermaßen verständlich.

Eine Sonderstellung nimmt der „Klammeraffe“ a ein. Als Baudot-Code wurde dafür die Buchstabenum-



schaltung verwendet. Dies ist von Vorteil, wenn die Fernschreibmaschine mit einer Motorabschaltautomatik ausgerüstet ist. Durch den Befehl LPRINT " a " kann die Maschine vom Mikrocomputer „aufgeweckt“ werden, ohne daß Zeichen verlorengehen.

Aus der Tabelle ist ersichtlich, daß die im Ziffern-Mode auszugebenden Zeichen mit gesetztem MSB tabelliert sind, also als negative Binärzahlen. Dadurch kann das Programm feststellen, ob die Ausgabe eines zusätzlichen Steuerzeichens (Ziffern- oder Buchstabenumschaltung) notwendig ist. Befindet sich die Fernschreibmaschine im Ziffern-Mode, so erhält das als Ziffern-Flag bezeichnete Byte einen von Null verschiedenen Wert.

Ein vom Programm kontrollierter Zähler sorgt dafür, daß nicht mehr als 66 Zeichen in eine Zeile geschrieben werden.

Das Treiberprogramm wird am bequemsten mittels eines einfachen BASIC-Programmes in einen reservierten RAM-Bereich geladen. Bild 5 zeigt das Listing dieses BPRINT genannten Programmes. Dieses Programm korrigiert auch einen leichten Fehler im Level-II-BASIC, durch den es manchmal zu Schwierigkeiten bei der Ausführung von READ-Anweisungen kommt (vgl. FUNKSCHAU 1979, Heft 10, Seite 582). Auch wird der Device Control Block für den Drucker entsprechend geändert. Der praktische Betriebsablauf mit diesem Programm sieht dann so aus:

1. Baudot-Interface mit dem TRS-80 verbinden und einschalten.
2. TRS-80 einschalten und auf die Frage MEMORY SIZE? mit 32560 antworten.

3. Das Programm BPRINT von der Kassette laden und laufen lassen.

4. Nach der Eingabe von NEW ist der TRS-80 mit Drucker betriebsbereit. Es ist dann nur noch gegebenenfalls der Motor der Fernschreibmaschine einzuschalten.

In der hier vorgestellten Form ist die Treiber-Software für einen TRS-80 mit 16K RAM gedacht. Bei größeren Speichern sollten die Programme entsprechend abgeändert werden. Da im Maschinenprogramm nur relative Sprünge verwendet werden, muß im Objektcode lediglich der Ladebefehl für das IX-Register geändert werden.

#### Stichworte zum Inhalt

Baudot-Fernschreiber, Treiberprogramm, serielle Ausgabe, Linienstrom

```

10 rem      bprint
20 rem basic - version      stand 13.5.79
30 rem von g. duddek
40 rem read - zaehler reparieren
50 poke 16553,255
60 rem 'device control block' aendern
70 poke 16421,2 : poke 16422,48 : poke 16423,127
80 rem treiberprogramm in den reservierten ram-bereich poken
90 for i=32560 to 32757
100 read a
110 poke i,a
120 next
130 end
140 data 245,221,229,213,221, 33,181,127,221, 86,252,121,254
150 data 13, 32, 14, 62,232,221,119,253, 62,226,221,119,254
160 data 22, 66, 24, 62,121,230, 63, 50, 86,127,221,126, 32
170 data 221,119,254,230,128, 32, 24,221,126,255,254, 0, 32
180 data 5,221,119,253, 24, 34, 62,255,221,119,253, 62, 0
190 data 221,119,255, 24, 22,221,126,255,254, 0, 40, 7, 62
200 data 0,221,119,253, 24, 8, 62,251,221,119,253,221,119
210 data 255,219, 3,230, 1, 32,250,221,126,253,254, 0, 40
220 data 8,211, 3,219, 3,230, 1, 32,250,221,126,254,211
230 data 3, 21, 40,151,221,114,252,209,221,225,241,201, 5
240 data 232, 4, 0, 31, 3, 25, 14, 9, 1, 13, 26, 20, 6
250 data 11, 15, 18, 28, 12, 24, 22, 23, 10, 5, 16, 7, 30
260 data 19, 29, 21, 17,153,153,153,153,153, 4,156,133,153
270 data 153,153,153,133,143,146,153,145,140,131,156,157,150
280 data 151,147,129,138,144,149,135,134,152,142,156,143,158
290 data 146,153, 4, 6, 53, 17, 19

```

Bild 5. BASIC-Programm zur Initialisierung des Drucker-Treibers



Bild 1. Steuerung eines externen Gerätes nach dem Memory-Mapped-System



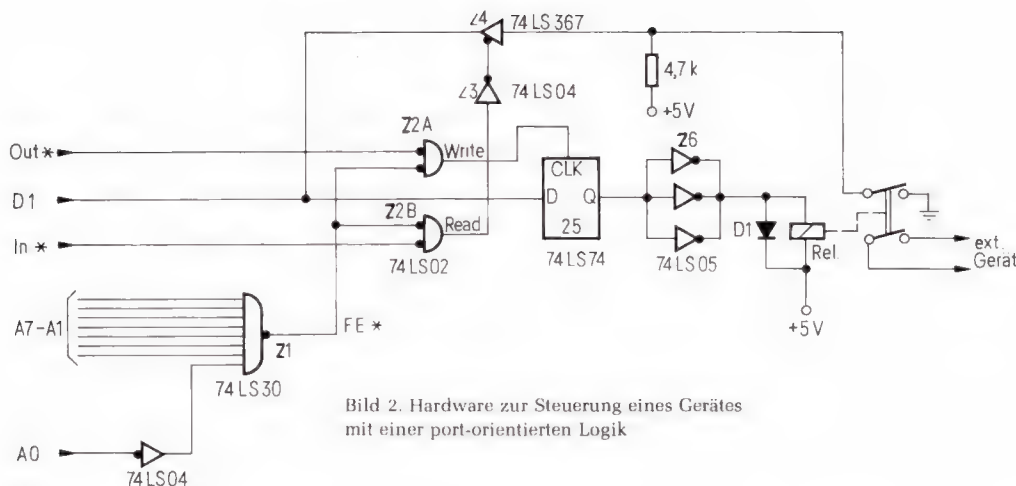


Bild 2. Hardware zur Steuerung eines Gerätes mit einer port-orientierten Logik

Wir wählen als Beispiel Port 255 (dezimal) entsprechend FF Hex. Die Assembler-Routine lautet:

OUT FF, 02 H; Einschalten des externen Gerätes

In Basic Level II würde es so aussehen:  
OUT 255,2

Bei einem Port-System verwenden wir also die Instruktionen IN und OUT und nur 8 Adreßleitungen für den Bereich 0000...00FF.

Soll man nun das Memory-Map- oder das Port-System verwenden? Das bleibt eigentlich jedem selbst überlassen. Entscheidet man sich für Memory-Map, dann sollten die I/O-Adressen so hoch wie möglich liegen, um Störungen mit dem Arbeitsspeicher zu umgehen. Wenn man das Portsystem benutzt, kann man 255 Adressenleitungen verwenden, weil der Kassettenrecorder den Port FF benutzt. Von der Hardware-Seite her bietet sich das Port-System eher an.

Warum? Es ist notwendig, extern eine Adressen-Decodierung vorzunehmen. Und während man beim Port-System nur 1 von 256 Möglichkeiten bei acht Adressenleitungen decodieren muß, fällt beim Memory-Map-System die Decodierung von 1 aus 65 536 Möglichkeiten bei 16 Adressenleitungen an.

### Das Memory-Mapped-System

Bild 1 zeigt die notwendige Hardware für ein Memory-Mapped-I/O-System. Die sechzehn Adressenleitungen werden hier mit NAND-Gattern decodiert. Mit einem Reed-Relais läßt sich ein externes Gerät steuern. Ein zweiter Relaiskontakt dient als Rückmeldung, ob auch tatsächlich ein Schaltvorgang stattfand.

Zusätzlich benötigt man eine selbständige 5-V-Stromversorgung, weil diese Schaltung nicht aus dem TRS-80 gespeist werden kann.

Ein kleines Beispielprogramm:

```
10 REM STEUERUNG VON EXTERNEN GERÄTEN
20 CLS: PRINT „SOLL DAS GERÄT EINGESCHALTET WERDEN“;
30 INPUT A$: IF A$ = „NEIN“ THEN GOTO 60
40 POKE 4092,2
50 B = 2 : A = PEEK (4092) : REM ANFORDERUNG NR. 3
60 END
```

In Zeile 40 schreibt die CPU das Binärwort 2 in den Speicher unter der Adresse 4092 (binär: 1000 1111 1111 1111). Die ICs Z1, Z2a und Z3 dekodieren die Adresse für dieses 16-bit-Wort, welches auch als Signal am Ausgang von Z3 anliegt. Dieses Signal gelangt an die Eingänge von Z2c und Z2d. Die POKE-Instruktion bedingt, daß der Write-Eingang der CPU zur gleichen Zeit „Low“ wird, wenn das 4092-Signal auf „low“ geht. Dabei wird am Ausgang von Z2c ein „High“-Signal frei, welches als WRITE-Signal interpretiert wird und an den Takt-Eingang des Z6 eines Latches (Zwischenspeicher) gelangt. Das Datenwort „2“, binär 0000

0010, gelangt als „High“-Signal über die Datenleitung D1 des Datenbus an den D-Eingang des Flipflops. Das Datenwort wird in Z6 abgespeichert. Das daraus resultierende „High“-Signal gelangt über den Ausgang Q von Z6 zu einem Relais Z7, dieser kann mit etwa 30 mA belastet werden. Während ein Kontakt das externe Gerät steuert, gibt der andere Kontakt ein Meldesignal über den Datenbus an den Computer zurück.

### Das Port-System

Bild 2 zeigt das Schaltbild des Port-orientierten Systems. Wie man sieht, ändert sich hardwareseitig nicht viel. Im Basic-Betriebsprogramm ändern wir 2 Zeilen:

```
40 OUT 254,2
50 B = 2 : A = INP 254
```

Selbstverständlich ist es auf ähnliche Weise auch möglich, nicht ein externes Gerät zu steuern, sondern periphere Logikzustände abzufragen. Der Experimentierfreudigkeit sind also kaum Grenzen gesetzt.

## Der AIM-65 im Amateurfunk

In den Heften 15/1979 und 16/1979 der FUNKSCHAU erschienen Programme für das Empfangen und Senden von Funkfern-schreiben sowie für das Erzeugen von Morsezeichen mittels der alphanumerischen Tastatur des AIM-65. Die Programme sind Beispiele dafür, daß man auch ohne flimmernden Bildschirm Texte verarbeiten kann und enthalten darüber hinaus typische Anwendungsbeispiele des VIA-Bausteins 6522, der in diesem Sonderheft näher erläutert wird.



Beim Mikrocomputer KIM-1 kann während des Austestens von Programmen nur sehr umständlich die Wirkung des zuletzt ausgeführten Befehls auf CPU-interne Register und andere Speicherzellen verfolgt werden. Dieser Beitrag zeigt, wie durch eine einfache Zusatzschaltung und ein Interruptprogramm mit nur einem Tastendruck der Inhalt von mehreren Speicherzellen angezeigt werden kann.

Dr. Rainer Gerlich

# Testhilfe für den KIM-1

## So funktioniert's

Das Monitorprogramm des KIM-1-Systems zeigt immer nur die Adresse einer Speicherzelle mit deren Inhalt an. Möchte man während des schrittweisen Programmtestens im Single-Step-Modus (SST-Modus) den Inhalt anderer Speicherzellen einblenden und das mühsame Eingeben neuer Adressen vermeiden, so besteht folgende Möglichkeit: Der Tester teilt durch ein geeignetes Signal mit, daß er entweder den Inhalt einer bestimmten Speicherzelle sehen oder in das Testprogramm an die Stelle des nächsten auszuführenden Befehls zurückverzweigen will. Solche Funktionen werden am besten durch einen Interrupt ausgelöst. Da das Monitor-Programm des KIM-1 das Interrupt-Disable-Bit im Statusregister setzt, kann dazu aber nur der NMI-Interrupt benutzt werden.

Im SST-Modus werden nur Befehlsfolgen, die im Bereich 1C00...1FFF stehen, ungehindert ausgeführt. In allen anderen Speicherbereichen wird nach der Ausführung eines Befehles ein NMI-Interrupt erzeugt, der eine Unterbrechung der Befehlsausführung und ein Verzweigen in das Monitor-Programm bewirkt. Damit das zur Testhilfe benötigte Interruptprogramm durchgehend abgearbeitet werden kann, ist daher ein kleiner Eingriff auf der KIM-Pla-

tine erforderlich. Außerdem muß ein Flag gesetzt werden, damit zwischen normalen „System“-Interrupts und den „Anzeigewechsel“-Interrupts unterschieden werden kann. Die dazu notwendigen Hardware-Änderungen zeigt Bild 1; das entsprechende Impulsdigramm geht aus Bild 2 hervor. Bild 3 zeigt die Änderungen auf der KIM-Platine.

Zwei Programmbeispiele, mit denen durch nur einen Tastendruck eine Adresse oder mehrere Adressen nacheinander ohne große Mühe angezeigt werden können, werden ebenfalls beschrieben. Zum Einsprung in das Interruptprogramm müssen die Speicherzellen 17FA und 17FB folgendermaßen besetzt werden: 17FA mit 80 und 17FB mit 17.

## Die Software

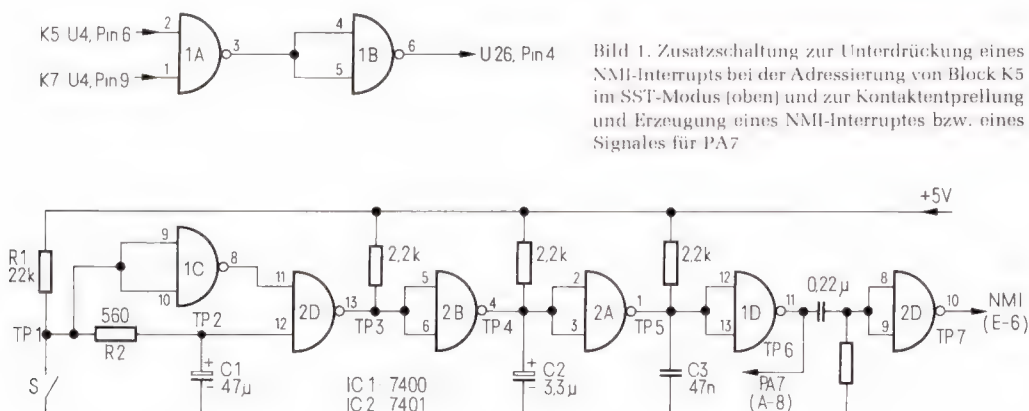
Aus der Vielzahl der Möglichkeiten sollen hier zwei Programme vorgestellt werden, die der einzelne Anwender leicht seinen Erfordernissen anpassen kann.

In beiden Programmen wird zunächst analysiert, ob der NMI-Interrupt vom Tester selbst oder von KIM ausgelöst wurde. Zur Unterscheidung wird Bit 7 von Port A (PA7) herangezogen. Hat PA7 nach der Verzweigung in das Interrupt-Programm den Zustand „L“, so

wird ein normaler Systeminterrupt angenommen und in das Monitor-Interrupt-Programm verzweigt.

Im Programm (Bild 4) wird bei jedem Test-Interrupt die Variable CNT inkrementiert und in Abhängigkeit von ihrem Wert eine Adresse in den Pointer 00FA/00FB (POINTL, POINTH) für die Display-Anzeige durch indizierte Adressierung mit dem X-Register geladen. Die Adresse und der Inhalt der Speicherzelle erscheinen dann in der Anzeige. Auf diese Weise kann der Tester sich nacheinander in einer von ihm gewählten Reihenfolge den Inhalt verschiedener Zellen ansehen. Unabhängig vom Interrupt-Programm kann mit Hilfe des Monitor-Programmes die Display-Adresse mit Hilfe der „+“-Taste weiter inkrementiert werden. Ist z. B. die erste anzuzeigende Adresse 00F1, so können durch Betätigen der „+“-Taste nacheinander auch die Inhalte der Register und des Akkumulators sichtbar gemacht werden. Bei Auslösen des nächsten Interrupts wird die nächste Adresse in den Display-Pointer geladen, danach kann wieder bei Bedarf die Adresse inkrementiert werden.

Liegt der Wert von CNT (17E4) nicht innerhalb des geforderten Bereiches (im Beispiel zwischen 0 und 9), so wird CNT auf 0 gesetzt. Hat CNT den Wert 0, so wird der momentane Wert von



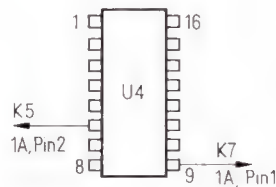


POINTL, POINTH nach 17E5, 17E6 (ADRL, ADRH) gerettet. Wenn CNT = 9 ist, wird die ursprüngliche Adresse wieder nach (POINTL, POINTH) übertragen und damit ins Testprogramm zurückverzweigt. Falls erforderlich, kann der Anwender die hier vorgegebene obere Grenze 9 abändern. Dazu ist die neue obere Grenze in die Speicherzelle 1798 anstelle von 09 einzutragen.

Bei Programmbeispiel 1 ist von Nachteil, daß immer alle vorgegebenen Stationen durchlaufen werden müssen, auch wenn momentan nur ein bestimmter Speicherbereich interessiert. Das wird in Beispiel 2 (Bild 5) vermieden. Bei diesem Programm kann der Taster durch eine Schalterstellung unter einer von mehreren Möglichkeiten auswählen. Beim ersten Interrupt wird die Basisadresse des gewünschten Speicherbereiches angezeigt und beim nächsten Interrupt erfolgt ein Sprung zurück in das zu testende Programm.

Da durch die vorgegebene Schalterstellung eine Prüfung auf zulässige Werte entfallen kann und somit mehr Speicherplatz zur Aufnahme des Interrupt-Programmes zur Verfügung steht, konnte bei diesem Programmbeispiel

Bild 3. Anschlußpunkte und Lage der Unterbrechungsstelle auf der KIM-1-Platine



KIM-1-Platine

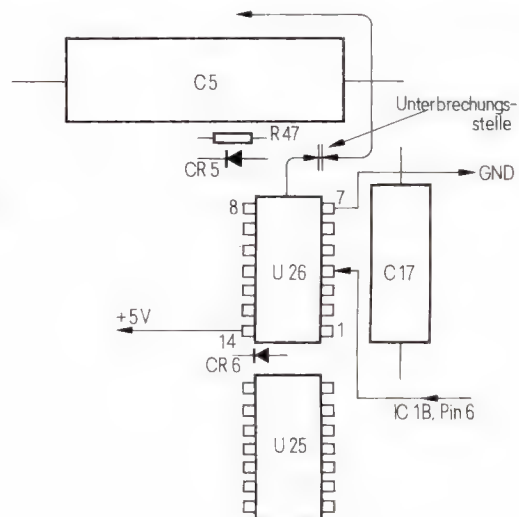


Bild 2. Zeitliches Verhalten der Schaltung in Bild 1

auch noch die Ausgabe über TTY realisiert werden.

Durch einen externen Schalter werden die Bits 0, 1, 2 des Ports A (PA0, PA1, PA2) modifiziert, so daß unter insgesamt 8 Basisadressen ausgewählt werden kann. Da die Arbeitsvariablen CNT, BASISL, BASISH aus Platzgründen nicht mehr im selben Speicherbereich wie das Programm untergebracht werden konnten, können diese Variablen im Unterschied zu Programmbeispiel 1 nicht mit dem Laden des Programmes gleichzeitig vorbesetzt werden. Bevor das Programm zum ersten Mal nach „Power on“ benutzt wird, ist daher eine Definition dieser Variablen notwendig. Dies gilt besonders für die Variable CNT, die vor dem ersten Programmstart unbedingt den Wert Null zugewiesen bekommen muß.

Verläuft der TTY-Test positiv, so wird der Inhalt der ausgewählten Speicherzelle ausgegeben und anschließend noch während des gleichen Interrupts die Verzweigungsadresse wieder geladen. Daher ist bei TTY-Anschluß ein weiterer Interrupt zum Verzweigen ins Testprogramm nicht erforderlich. Zur TTY-Ausgabe werden die Systemprogramme PRTPNT, PRTBYT, OUTSP, CRLF des KIM-1 verwendet [1].

Möchte der Benutzer mehr oder weniger Möglichkeiten zur Auswahl haben, so müssen dazu die Speicherzellen 1784 (Definition des Data Direction Registers für Port A) und 1797 (Maske für eingelesenes Byte) abgeändert werden. Die Adressen der Speicherzellen, deren Inhalt angezeigt werden soll, sind für

Programm 1 von 17D2 bis 17E3, für Programm 2 von 00DC bis 00EB einzutragen.

### Erforderliche Hardware-Änderungen

Beim KIM-1 wird im SST-Modus mit Hilfe des SYNC-Signals, das das Laden des nächsten Operationscodes ankündigt, ein NMI-Interrupt erzeugt [2]. Die fortlaufende Bearbeitung des Operationscodes wird dann nach Ausführung eines Befehls immer unterbrochen. Ausgenommen davon sind Programme im Bereich 1C00...1FFF (Block K7). Jedesmal, wenn eine Zelle in K7 adressiert wird, nimmt ein Eingang des NAND-Gatters U26, B den Low-Zustand an, so daß dann ein Interrupt verhindert wird. Um die ungehinderte Ar-



beit des Interrupt-Programmes für die Testunterstützung im SST-Modus zu ermöglichen, muß bei der Adressierung des Speicherbereiches, in den dieses Interrupt-Programm geladen wird, ebenfalls der SYNC-NMI-Interrupt unterdrückt werden.

Als Speicherbereich für das Interrupt-Programm bietet sich das vom KIM-1 nicht genutzte RAM auf den beiden Chips 6530 (U2 und U3 auf der KIM-Platine) an, d. h. der Speicherbereich 1780 bis 17E6. Dieses RAM ist der einzige (freie) Speicher im Block K5, so daß durch eine Sonderstellung dieses Blockes trotzdem andere Programme im übrigen Speicher weiter schrittweise ausgetestet werden können. Eine direkte „Wired-OR“-Verbindung zwischen K5 und K7 ist leider nicht möglich, da beide Signale entkoppelt bleiben müssen. Daher sind zwei NAND-Gatter zusätzlich erforderlich. Durch einen Taster wird nach Kontaktentprellung ein Ausgangsimpuls für PA7 und ein Interrupt-Impuls erzeugt.

```

1780 48      PHA
1781 A9 7F    LDA =7F
1783 2D 01 17 AND 1701
1786 8D 01 17 STA 1701
1789 AD 00 17 LDA 1700
178C 30 04     BMI 1792
178E 68      PLA
178F 4C 00 1C JMP 1C00
1792 8A      TXA
1793 48      PHA
1794 AD E4 17 LDA 17E4
1797 C9 09     CMP =09
1799 D0 0D     BNE 17A8
179B AD E5 17 LDA 17E5
179E 85 FA     STA FA
17A0 AD E6 17 LDA 17E6
17A3 85 FB     STA FB
17A5 4C C9 17 JMP 17C9
17A8 90 05     BCC 17AF
17AA A9 00     LDA =00
17AC 8D E4 17 STA 17E4
17AF C9 00     CMP =00
17B1 90 F7     BCC 17AA
17B3 0A      ASL A
17B4 AA      TAX
17B5 D0 0A     BNE 17C1
17B7 A5 FA     LDA FA
17B9 8D E5 17 STA 17E5
17BC A5 FB     LDA FB
17BE 8D E6 17 STA 17E6
17C1 BD D2 17 LDA 17D2,X
17C4 85 FA     STA FA
17C6 BD D3 17 LDA 17D3,X
17C9 85 FB     STA FB
17CB EE E4 17 INC 17E4
17CE 68      PLA
17CF AA      TAX
17D0 68      PLA
17D1 40      RTI

```

Bild 4.  
Programmbeispiel 1 mit Display-Ansteuerung

Beim Schließen des Schalters S wird am Ausgang von Gatter 2D ein Normimpuls erzeugt, dessen Länge vom Ladezustand von Kondensator C1 abhängt. Da C1 sich beim Schließen von S relativ schnell über R2 entlädt, aber nur langsam wieder (z. B. beim Prellen von S) durch R1 aufgeladen werden kann, ändert sich der Ausgangszustand von Gatter 2D bis auf die Einschwingphase nicht, da ein Eingang auf „L“ liegt, bis Schalter S wieder geöffnet wird. Nur während des kurzen Entladevorgangs beim Schließen von S kann sich das Kontaktprellen auf den Ausgang von Gatter 2D auswirken. Kondensator C2 kann jedoch erst dann voll aufgeladen werden, wenn der Schalter S eine längere Zeit nicht geöffnet ist. Daher wird am Ausgang von Gatter 2A bereits ein relativ sauberer Impuls erzeugt. Kondensator C3 integriert die verbliebenen, geringen restlichen Störungen weg. Nach Invertierung gelangt der Impuls zu PA7 (Application Connector, Pin 8) und nach Differenzierung an den NMI-Eingang (Expansion Connector, Pin 6). Das differenzierte Signal erzeugt einen NMI-Interrupt, das PA7-Signal steuert die Verzweigung innerhalb des Interrupt-Programms in den Teil, der den Anzeigewechsel bewirkt.

Die Länge des NMI-Impulses beträgt ca. 35 µs, der PA7-Impuls ist immer länger als 3 ms. Die maximale Wiederholungsrate für einen Testhilfe-Interrupt wird durch das Zeitglied R1/C1 bestimmt; sie liegt max. bei etwa 3 Hz. Wird der Schalter S öfter pro Zeiteinheit geschlossen, so wird kein Ausgangs-Impuls erzeugt.

Beim Anschließen des KIM muß die Verbindung von U26, Pin 4 zu U4, Pin 9 unterbrochen werden. Am besten geschieht dies durch Auftrennen der Leiterbahn, die zwischen Pin 7 und 8 unter U26 hervortritt. Da von U4, Pin 9 noch eine Verbindung zu U2, Pin 4 besteht, sollte die Leiterbahn nicht in der Nähe von U4 unterbrochen werden. GND und + 5 V können von Pin 7 bzw. Pin 14 von U26 (7438) abgenommen werden. Die Zusatzschaltung kann leicht auf einer kleinen Platine über der KIM-Platine angebracht werden.

### Ausblick

Es soll nicht unerwähnt bleiben, daß auf diese Weise nicht nur Interrupt-Programme für die Speicherinhaltsanzeige realisiert werden können, sondern auch andere Testunterstützungen, z. B. die (Neu-)Definition von Variablen vor einem Testschritt durchgeführt werden können. Dazu aber noch ein Hinweis: Beim Behandeln eines Interrupts müssen Akkumulator und Register, falls sie im Interrupt-Programm benutzt werden, auf den Stack gerettet und vor dem Rücksprung wieder vom Stack geholt werden.

### Literatur

- 1 KIM- und SYM-Monitor-Unterprogramme. FUNKSCHAU 1979, Heft 11, Seite 652
- 2 KIM-1 Users' Manual. MOS Technology Inc., 1976

### Stichworte zum Inhalt

Single-Step-Modus, SST, KIM-1, 6502, Entprellung, Interrupt, Registeranzeige.

```

1780 48      PHA
1781 8A      TXA
1782 48      PHA
1783 A9 78    LDA =78
1785 2D 01 17 AND 1701
1788 8D 01 17 STA 1701
178B AD 00 17 LDA 1700
178E 30 06     BMI 1796
1790 68      PLA
1791 AA      TAX
1792 68      PLA
1793 4C 00 1C JMP 1C00
1796 29 07     AND =07
1798 0A      ASL A
1799 AA      TAX
179A 98      TYA
179B 48      PHA
179C 24 EC     BIT EC
179E 30 37     BMI 17D7
17A0 C6 EC     DEC EC
17A2 A5 FA     LDA FA
17A4 85 ED     STA ED
17A6 A5 FB     LDA FB
17A8 85 EE     STA EE
17AA B5 DC     LDA DC,X
17AC 85 FA     STA FA
17AE B5 DD     LDA DD,X
17B0 85 FB     STA FB
17B2 A0 00     LDY =00
17B4 8C 41 17 STY 1741
17B7 A9 3F     LDA =3F
17B9 8D 43 17 STA 1743
17BC 8C 42 17 STY 1742
17BF A9 01     LDA =01
17C1 2C 40 17 BIT 1740
17C4 D0 1B     BNE 17E1
17C6 20 2F 1E JSR 1E2F
17C9 20 1E 1E JSR 1E1E
17CC 20 9E 1E JSR 1E9E
17CF B1 FA     LDA (FA),Y
17D1 20 3B 1E JSR 1E3B
17D4 20 9E 1E JSR 1E9E
17D7 E6 EC     INC EC
17D9 A5 ED     LDA ED
17DB 85 FA     STA FA
17DD A5 EE     LDA EE
17DF 85 FB     STA FB
17E1 68      PLA
17E2 A8      TAY
17E3 68      PLA
17E4 AA      TAX
17E5 68      PLA
17E6 40      RTI

```

Bild 5. Programmbeispiel 2 mit TTY-Ausgabe



# Tastatur- und Kassettenrecorder-Interface

Die hier vorgestellte Schaltung dient als Interface zwischen einer Tastatur (beim Autor: Datamega 7204) und einem Datensichtgerät (Selbstentwurf und Selbstbau, angeregt durch ELEKTRONIK 1,2/77), sowie als Interface zwischen diesem Sichtgerät und einem Kassetten-Recorder. Außerdem ist das Senden und der Empfang von seriellen digitalen Daten möglich. Die Geschwindigkeit beträgt dabei – wie auch beim Datenverkehr mit einem Tonbandgerät – 110 Baud. Sie läßt sich aber bis 300 Baud steigern, wenn bestimmte Änderungen vorgenommen werden können. Der Einsatz in MP-Systemen ist ebenfalls denkbar. Die Verbindung kann dabei über die seriellen Ein-/Ausgänge oder über den 8-bit-parallel-Datenbus erfolgen.

Das Interface arbeitet mit einem Strobe-Signal. Die Steuerung der Parallel-Serien- bzw. Serien-Parallel-Wandlung sowie die Generierung von Start- und Stop-Bits besorgt eine UART-Schaltung TMS 6011. Eine Reihe von Möglichkeiten, die diese MOS-LSI-Schaltung bietet, bleiben hier ungenutzt: So werden die Fehler-Flags nicht verwendet, auch auf die Erzeugung des Paritäts-Bits wurde verzichtet.

Den Takt für das IC 6011 liefert ein mit einem Timer-IC 555 aufgebauter Oszillator. Die Umwandlung der seriellen Daten-Folge in ein FSK-Signal besorgt eine in FUNKSCHAU 15/78 beschriebene Modem-Schaltung. Der Empfang von FSK-Signalen erfolgt mit einer in der FUNKSCHAU Nr. 14/78 vorgestellten Demodulator-Schaltung.

Alle digitalen Ausgänge des Interface sind offene Kollektor-Ausgänge. Der Datenbus ist bidirektional ausgelegt.

## Funktionsweise

Das Interface verfügt über zwei Betriebsarten, die mittels hex 0E und 0F als Tastatur-Befehle wählbar sind: Senden und Empfang. Eine mit Germanium-Dioden aufgebaute Matrix und eine Nand-Verknüpfung besorgen die Decodierung der Befehlsörter. Das Monoflop MF 1 gibt die Decodierung erst frei, wenn das Strobe-Signal der Tastatur erzeugt ist. Damit werden Fehlschaltungen durch noch nicht stabilisierte Tastatur-Daten verhindert. Ein aus zwei Nand-Gattern aufgebautes Flipflop FF 1 speichert die gewählte

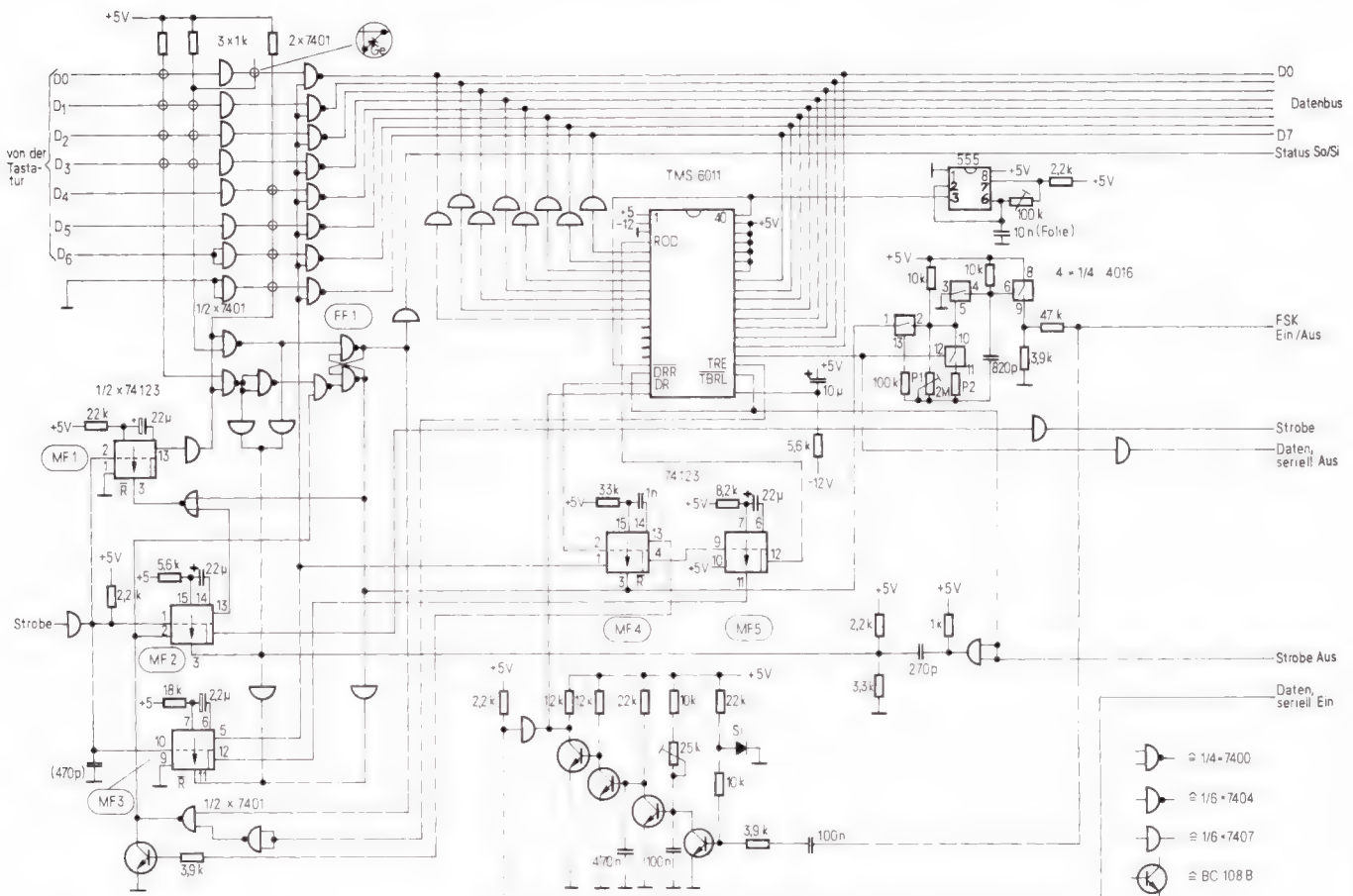


Bild 1. Gesamtschaltbild des Interface für eine ASCII-Tastatur mit parallelen Ausgängen



Betriebsart und steuert den Status-SO/SI-Ausgang: „H“ zeigt die Betriebsart „Senden“ (SO = shift out) an.

Die Verknüpfung der Decoder-Ausgänge mit den Reset-Eingängen von MF 2 (Strobe-Monoflop) und MF 3 (Tastatur-Daten-Übergabe-Steuerung) sorgt dafür, daß bei den Datenwörtern OE und OF kein Strobe erzeugt wird. Alle übrigen Datenwörter, die von der Tastatur erzeugt werden, gelangen auf den Datenbus. Die Übergabe steuert MF 3, das mit der positiven Flanke des Tastatur-Strobe-Signals getriggert wird. Mit seiner negativen Flanke triggert das Strobe-Signal der Tastatur das Strobe-Monoflop MF 2. Mit der positiven Flanke des Q-Ausgangs von MF 2 wird MF 1 bei Betriebsart Empfang zurückgesetzt, damit bei schneller Tastatur-Bedienung keine Fehlschaltungen erfolgen.

Die Tastatur-Daten bleiben solange auf den Datenbus aufgeschaltet, solange der Q-Ausgang von MF 3 auf „H“ liegt, längstens also solange, wie durch die Zeitglieder von MF 3 festgelegt ist. Auch das Strobe-Signal bleibt längstens für die Zeit auf „L“, die durch die Zeitkomponenten von MF 2 determiniert ist. MF 2 und MF 3 werden mit jeder positiven Flanke von „Strobe aus“ zurückgesetzt. Dieses Signal wird aus dem Strobe-Signal abgeleitet, indem es mit einer der Zugriffszeit der verwendeten Halbleiterspeicher entsprechenden Verzögerung auf „L“ geht und für etwa 10 µs in diesem Zustand verharrt, wenn nicht Bildschirmbefehle decodiert werden, zu deren Ausführung eine gewisse Zeit benötigt wird (Löschen des gesamten Bildinhalts, von einzelnen Zeilen oder Bildschirmteilen). Für diese Fälle müssen die betreffenden Befehle und das Strobe-Signal maximal 2 Bilddurchläufe (40 ms) stabil sein. Erst nach Ausführung der Befehle geht „Strobe Aus“ auf „H“ zurück. Andererseits soll aber die Zeit, die alle anderen Daten auf dem Datenbus aufgeschaltet sind, möglichst kurz sein, damit ein störendes Flimmern des Bildschirms vermieden wird. Es sei noch erwähnt, daß man selbstverständlich an die Monoflop-Zeiten von MF 2, 3 und auch von MF 5 nicht gebunden ist. Sie können ganz vom konkreten Einsatzzweck des Interfaces abhängig gemacht werden. Es ist jedoch darauf zu achten, daß sie nicht länger als die für die Übertragung eines Datenworts an die seriellen Ein-/Ausgänge benötigten Zeiten sein sollen.

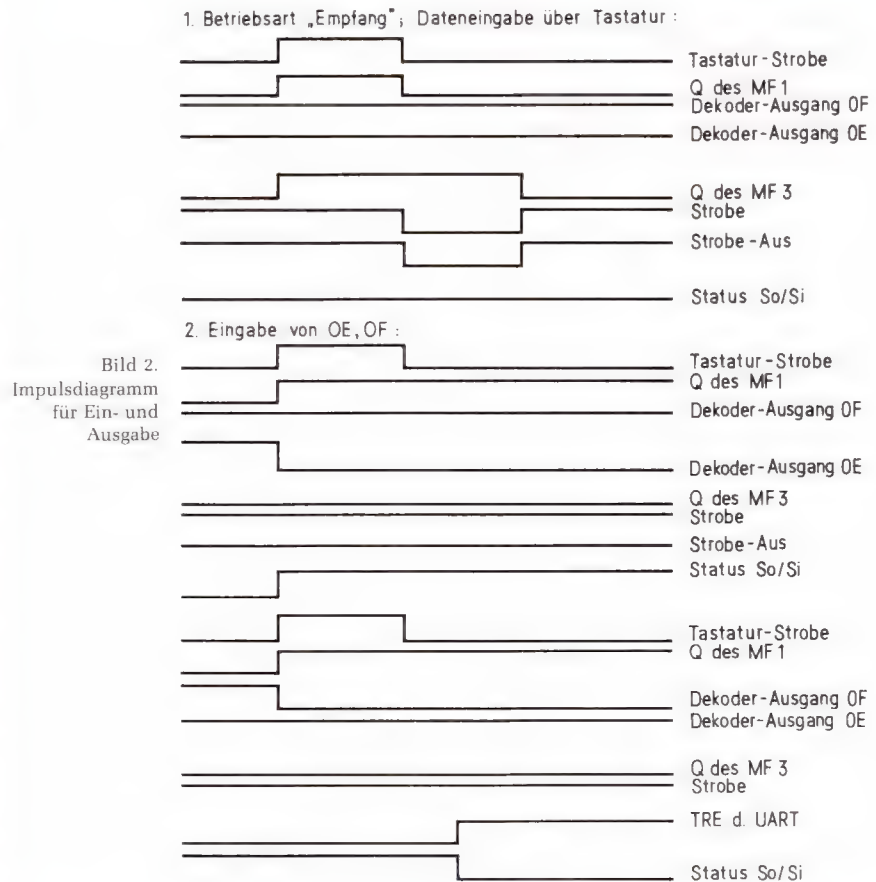
Bei der Betriebsart „Empfang“ ist neben der Übergabe von Tastatur-Daten auch der Empfang von auf Kassette gespeicherten Daten möglich. Gesteuert durch den Ausgang „DR“ des UART triggert MF 4 das Strobe-MF 2 und erzeugt so „L“ auf der Strobe-Leitung, sobald serielle Daten vollständig empfangen wurden. Um die Monoflop-Zeit von

MF 4 früher schaltet MF 5 die empfangenen Daten auf den Datenbus. Das geschieht über den Eingang „ROD“ des UART. Die Verknüpfung von Ausgängen der MFs 2 und 3 mit Eingängen MFs 4 und 5 sorgt dafür, daß die Tastatur-Eingabe Vorrang hat und die auf den Datenbus aufgeschalteten Datenwörter eindeutig und fehlerfrei sind.

Der Empfang von seriellen Daten ist möglich, wenn dafür gesorgt wird, daß Pin 20 des UART „H“ ist, wenn der Eingang „Daten seriell ein“ „H“ ist.

wird, beginnt die automatische Übergabe von Daten an das Tonbandgerät. Der Ausgang TRE (Pin 24) des UART fordert über MF 2 und Strobe ein neues Datenwort an, das bei „Strobe aus“=„L“ in den UART übernommen wird.

Am Ausgang „Daten seriell aus“ steht die serielle Daten-Folge zur Verfügung. Dies gilt auch, wenn auf Betriebsart „Empfang“ geschaltet ist. Dann erfolgt eine Ausgabe jedoch nur, wenn vorher ein Datenwort über Ta-



Erzeugt die Tastatur den Befehl hex 0E, kippt das FF 1 und „status-SO/SI“ geht auf „H“. Die Übergabe von Tastaturdaten auf den Datenbus wird durch Reset des Monoflops 3 gesperrt. Durch Reset des MFs 4 wird eine Triggerung von MF 5 verhindert. So bleiben die Daten-Ausgänge des UART bei der Betriebsart „Senden“ gesperrt. Das Modem wird freigegeben, was sich über den Lautsprecher eines angeschlossenen, aufnahmebereiten Kassetten-Recorders verfolgen läßt. Hörbar wird der Ton, der Digital-„H“ codiert. Eine automatische Ausgabe von Daten erfolgt noch nicht. Der Start der Ausgabe eines Datenwortes wird mit der positiven Flanke von „Strobe aus“ bewirkt. Da wegen Reset des MF 2 bei decodiertem OE-Befehl (Sendebehl) kein Strobe erzeugt wurde, unterblieb auch „Strobe aus“. Erst wenn eine weitere beliebige Taste – außer hex 0E und 0F – betätigt

statur, den seriellen Daten-Eingang oder den FSK-Eingang eingegeben wurde. Eine automatische Anforderung über TRE unterbleibt, da TRE bei der Betriebsart „Empfang“ MF 2 nicht triggern kann.

Bei der Rückschaltung von „Senden“ auf „Empfang“ muß sichergestellt sein, daß zuvor ein Datenwort vollständig gesendet wurde. Dies geschieht durch die Verknüpfung zwischen dem Dekoder-Ausgang für Befehl hex 0F und dem FF 1. Das Flipflop kann nur dann kippen, wenn der UART durch eine positive Flanke am Eingang B des MF 2 (Pin 2) signalisiert, daß ein neues Datenwort übertragen werden kann. Damit ist gewährleistet, daß das FF 1 synchronisiert schaltet, obwohl der Umschaltbefehl i.d.R. asynchron in Bezug auf den Sendevorgang erfolgt. In der Betriebsart Senden sorgt das mit dem FF 1 verknüpfte Nand-Gatter am Re-



set-Eingang von MF 1 (Pin 3) dafür, daß MF 1 durch „H“ am Q-Ausgang von MF 2 nicht zurückgesetzt werden kann. Daher bleibt ein eventueller Befehl hex 0F (oder hex 0E) an den Dekoder-Ausgängen für die volle Monoflop-Zeit von MF 1 wirksam. Diese Zeit muß mindestens so lang sein, wie die vollständige Übertragung eines Daten-Wortes auf Tonband (oder über den seriellen Digitalausgang) dauert. Bei einer Geschwindigkeit von 110 Baud ergibt sich, daß 100 ms ausreichend sind, was die Dimensionierung der Zeitglieder von MF 1 sicherstellt.

### Einstellarbeiten

Die Frequenz des Taktoszillators (NE555) für den UART ist auf 16fache Übertragungsgeschwindigkeit einzustellen. Bei 110 Baud ergeben sich 1760 Hz. Hochwertige Bauteile sind sehr zu empfehlen! Mit P 1 wird der FSK-Modulator auf 1275 Hz eingestellt (bei „L“ an Pin 12 des 4fach-Analog-Schalters

4016 und Betriebsart Senden). Bei „H“ an Pin 12 des 4016 ist mit P 2 auf 2125 Hz einzuregulieren. Auch hier empfehlen sich hochwertige Bauteile für die frequenzbestimmenden Glieder!

Der FSK-Demodulator wird mit dem 25-k $\Omega$ -Trimmer auf störungsfreien Empfang eingestellt (vgl. FUNK-SCHAU 1978, Heft 14). Der Anschluß an das Tonbandgerät geschieht über den Überspieleingang und eine Diodenbuchse im Interface.

Steht ein dem „Strobe-aus“-Signal entsprechendes Signal nicht zur Verfügung, läßt es sich ggf. aus den vorhandenen Signalen erzeugen, zumal bei der Interface-Schaltung ein Monoflop (1/2 74123) ungenutzt bleibt. Möglicherweise kann auch ganz darauf verzichtet werden und „Strobe aus“ mit „Strobe“ verbunden werden. Der Kondensator 470 pF an den Eingängen der MFs 1, 2 und 3 dient zur Reduktion von Störungen. Er erwies sich beim Autor als sehr wirkungsvoll.

Alle verwendeten Transistoren sind unkritische NPN-Typen (BC 108, BC 170 o. ä.). Alle offenen Kollektor-Ausgänge wurden mit Pull-up-Widerständen in der Größenordnung 1 k $\Omega$  versehen, sofern nichts anderes angegeben.

Die acht Treibergatter zwischen den Datenausgängen des UART und dem Datenbus können entfallen, wenn die Datenbusanschlüsse mit nicht mehr als einer TTL-Last belastet werden. Schließlich können auch die Treibergatter in den Steuerbusanschlüssen des Interfaces entfallen, wenn auf die Möglichkeit der wired-or-Verknüpfung verzichtet werden soll.

Die gesamte Schaltung fand auf einer Europa-Karte 100 mm x 160 mm Platz und wurde als Einschub für ein Veroboard-19“-Gehäuse ausgeführt. Es wurde keine Platine geätzt, sondern eine Art Fädeltechnik angewandt. Eine 31polige DIN-Leiste stellt die Verbindung zum Daten- und Kontrollbus des Sichtgerätes her.

## Kassetten-Chaos

Nach wie vor sind Tonband-Kassetten das am weitesten verbreitete Speichermedium für Mikrocomputer. Da sich nach einiger Zeit eine ganze Menge von Programmen anhäuft, sollte man sich frühzeitig überlegen, wie man dafür sorgt, daß man ein Programm auch wiederfindet. Es gibt authentische Beispiele, daß ein PET-Besitzer ein langes BASIC-Programm neu schreiben mußte, weil er es in seinem Kassetten-Chaos nicht mehr gefunden hatte...

Der erste Schritt zur Vermeidung dieses Chaos ist die Verwendung eines Kassettenrecorders mit Bandzählwerk. Solche Recorder sind bereits um 100 DM zu haben; der PET 2001 besitzt leider kein Zählwerk, ein nachträglich einbaubares elektronisches Bandzählwerk wurde aber bereits in FUNK-

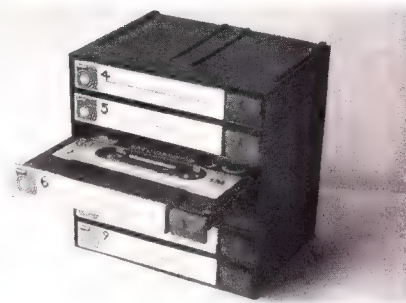
SCHAU 1978, Heft 21, und 1979, Heft 7, beschrieben, so daß sich dieses Problem sicher lösen läßt.

Die Methode, Kassetten mit nur wenigen Minuten Laufzeit zu verwenden und so nur ein Programm pro Kassette aufzuzeichnen, ruft schon bei einigen zehn Programmen Platzprobleme hervor. Besser ist es daher, Kassetten mit 2 x 30 min Laufzeit zu verwenden (C 60). C-120-Kassetten besitzen erfahrungsgemäß eine schlechtere mechanische Qualität und verursachen wegen ihrer häufigeren Drop-Outs oft Probleme beim Einlesen der Programme.

Zum Aufbewahren der Kassetten eignen sich recht gut die z. B. von BASF jetzt durchweg verwendeten, stapelbaren schubladenartigen Fächer des Typs „C-Box“. Diese Kassettenfächer sind auch ohne Inhalt recht preiswert erhältlich. Am besten numeriert man ihr Beschriftungsfeld mit 1 beginnend in aufsteigender Folge, ebenso wie die Kassetten selbst.

Jetzt hat man zwar einige durchnummerierte Kassetten, weiß aber noch nicht, was z. B. auf Kassette 2 für Programme aufgezeichnet sind. Hier gibt es nun zwei Möglichkeiten: Die erste und einfachste ist, sich einfach eine Liste zu schreiben und an die Wand zu hängen. Die zweite ist unvergleichlich besser, setzt aber das Vorhandensein eines Druckers oder Fernschreibers als Ausgabemöglichkeit für den Mikrocomputer sowie ein Text-Editor-Programm voraus. Dafür eignet sich z. B. „KIM auf Datensuche“ aus FUNK-

SCHAU 1978, Heft 24, oder auch der im AIM-65 vorhandene Editor. In jede Zeile kann man dann Kassetten-Nummer, Bandzählwerks-Stand, Identifikationszahl bzw. File-Name, Startadresse und Titel des Programms eintragen. Dieses Verfahren hat den Vorteil, daß man z. B. auf Kassette Nr. 1 ein Inhaltsverzeichnis speichern und bei Bedarf



ausdrucken kann, das sich ohne weiteres ändern läßt; enthält z. B. von 10 Kassetten die Nr. 4 nun beim Zählwerksstand 028 ein neues Programm, so läßt sich die entsprechende Zeile leicht mit dem Editor einfügen, ohne von Hand die komplette Liste neu schreiben zu müssen.

Noch ein Tip: Nehmen Sie jedes Programm wenigstens zweimal nacheinander auf. Kassetten sind kein „sicheres“ Speichermedium, und schon ein Tropfen Benzin macht ein ganzes Programm „unlesbar“. Wer ganz sicher gehen will, speichert ein Programm auf mehreren Kassetten. Fe.





In Verbindung mit einem universellen Hardware-Interface, das mit den Anschlußleisten des PET 2001 verbunden wird, lassen sich per BASIC Logikzustände überwachen, Telefonnummern wählen, Entfernungen berechnen, Antennenrotoren steuern, Temperaturen messen und Roulette spielen.

# PET, der Alleskönner

## Hardware

Das Universal-Interface verfügt über drei Anschlußleisten, die direkt an die PET-Leisten passen, wenn man geeignete Steckverbinder an sie lötet. Die PET-User-Ports stehen dann an Lötstiften zur Verfügung. Ein Lautsprecher gestattet die Reproduktion von Musik und Geräuschen, und ein Video-Ausgang (Synchronfrequenz allerdings 60 Hz) erlaubt den Anschluß eines externen Monitors. Auf die Wiedergabe der Schaltung wurde hier verzichtet. Bild 1 zeigt die Lötseite der Platine, Bild 2 die Bestückungsseite.

Zur Hardware-Simulation externer Geräte dient die „Port-Control“-Platine, deren Schaltung Bild 3 zeigt. Das Platinenlayout ist in Bild 4 und die Bestückung in Bild 5 wiedergegeben. Mit den Schaltern lassen sich definierte Logikzustände an die Ports anlegen, und die LEDs dienen zur Anzeige von Low oder High. Für die Verwendung als Telefon-Wählautomat wurde ebenfalls eine Schaltung entwickelt (Bild 6); Bild 7 zeigt eine passende Platine. Bild 8 ihren Bestückungsplan. Eine letzte Schaltung (Bild 9) dient der Temperaturüberwachung; hier wurde keine Platine entwickelt. Diese Schaltung verwendet den IEC-Bus und zeigt, wie sich diese nützliche Einrichtung des PET verwenden läßt.

## Software

### PET als Schaltuhr

Die Interface-Platine ist an die acht User-Port-Leitungen und mit der Versorgungsspannung zu verbinden. Nach dem Starten des Programms kann man die Uhrzeit sechsstellig und die Schaltzeiten vierstellig eingeben; mit der Space-Taste kann man zu einer anderen Zeit, mit „Return“ zu einem anderen Steuerausgang springen. Jeder Ausgang des User-Port kann ein Gerät steuern, insgesamt also acht. Bild 10 zeigt das Programm-Listing.

### Telefoncomputer

Das Programm in Bild 11 verwaltet Ihr Telefonregister nach folgenden Such-Gesichtspunkten: Numerierung (Data 1 bis XXXXX), Namen, Kennzeichen (z. B. Beruf, Firma usw.) oder Handruf, d. h. die zu wählende Num-

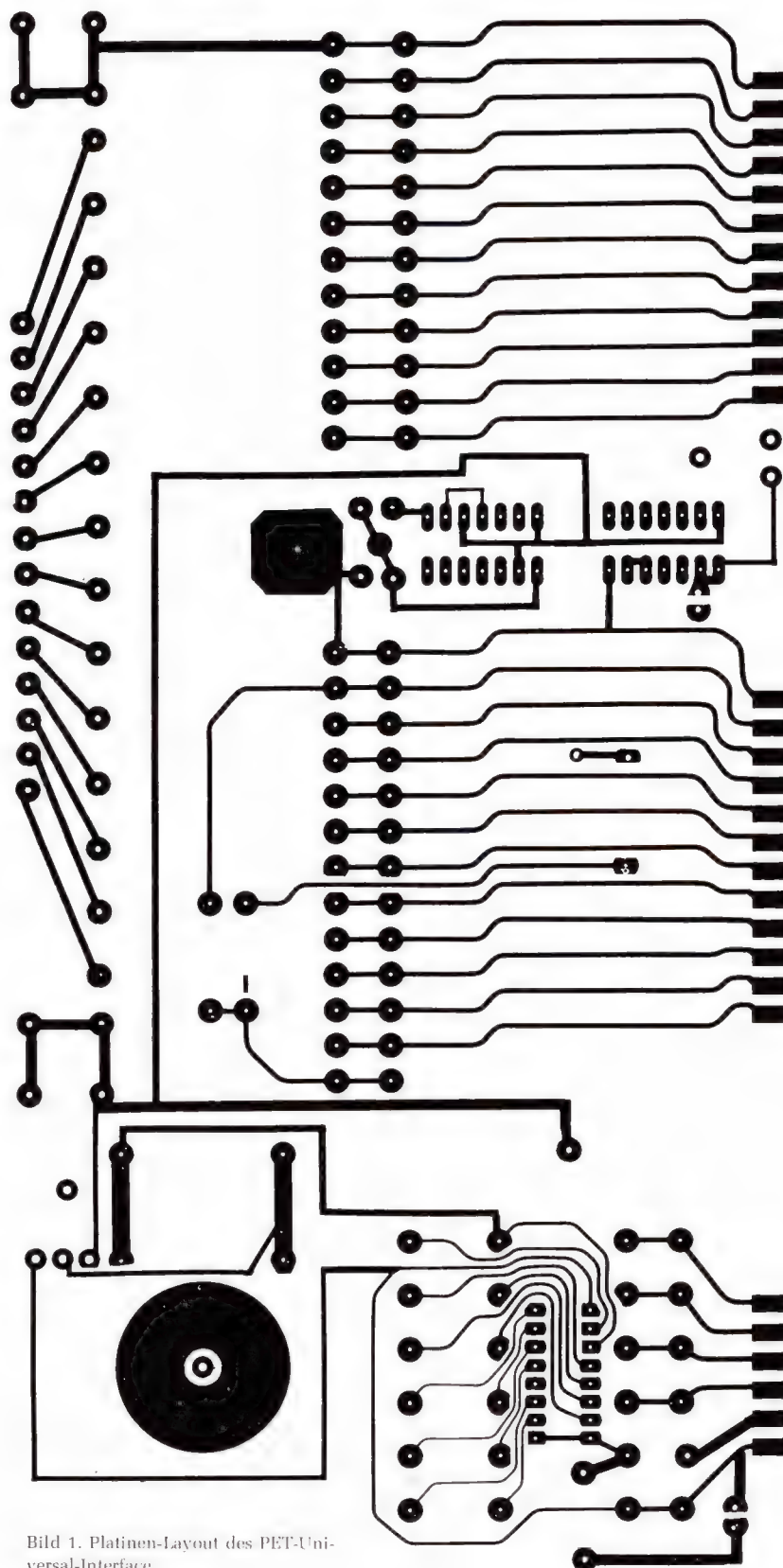


Bild 1. Platinen-Layout des PET-Universal-Interface



mer wird direkt eingetippt. Bitte beachten Sie, daß die Verwendung dieses Telefoncomputers in Deutschland nicht an Amtsleitungen zulässig ist! Der mit E gekennzeichnete Punkt des Telefon-In-

terface muß mit Bit 1 des IEC-Bus verbunden werden. Wenn die LED leuchtet, ist der Wählkontakt geschlossen. Beim Wählen muß natürlich der Hörer des Fernsprechers abgehoben sein. Die

Namen und Telefonnummern müssen als DATA in den Zeilen 1000...2000 abgelegt sein.

#### Entfernungsberechnung

Den Funkamateuren unter unseren Lesern errechnet das Programm von Bild 12 die Entfernung zwischen zwei sog. QTH-Kennern. Das Hardware-Interface wird hier nicht benötigt. Die Bedienung geht aus dem Programmablauf eindeutig hervor, so daß wir uns hier nähere Einzelheiten ersparen können. (Ein ähnliches Programm ist in diesem Heft auch für den TRS-80 abgedruckt.)

#### Rotorsteuerung

Dieses Programm (Bild 13) eröffnet Funkamateuren neue Dimensionen in der Abwicklung des Funkverkehrs mit drehbaren Antennen. Es enthält eine Kartei mit Rufzeichen, Namen, Wohnorten, Entfernungen und Antennenrichtungen. Der Rotor wird von den Ausgängen 1...4 des IEC-Bus gesteuert und dreht sich automatisch in die Richtung. Auf eine detaillierte Beschreibung der Hardware wurde hier absichtlich verzichtet, da sich die Dimensionierung nach der Rotor-Elektronik richtet.

#### Roulette

Für das Roulette-Spiel (Bild 14) werden 16 LEDs benötigt, die über einen Binärdecoder (74 L 154) an die Anschlüsse 1...4 des IEC-Bus angeschlossen werden. Das Programm wird mit RETURN gestartet und gestoppt; die „Kugel“ rollt dabei langsam aus, und die Gewinnzahl erscheint auf dem Bildschirm. Das Roulette-Interface eignet sich auch zur Verwendung mit der Rotor-Steuerung (s. o.), und der LED-Kreis zeigt dann die Rotorstellung an.

#### Temperaturüberwachung

Das Programm in Bild 15 tut eigentlich viel mehr, als nur die Temperatur zu regeln; es überprüft auch den Wasserstand und die Funktionsfähigkeit von Temperatur- und Niveaufühlern. Einzelheiten gehen aus dem Programm selbst hervor; mit leichten Änderungen läßt es sich für vielfältige Steuer-, Regel- und Überwachungsaufgaben einsetzen.

#### Port-Control

Bild 16 zeigt eine einfache Möglichkeit, die I/O-Zustände übersichtlich auf dem PET-Bildschirm darzustellen. Es arbeitet mit dem Port-Control-Interface. Bild 16 zeigt schließlich ein Programm zur Temperatur-Überwachung.

Willi Billen  
Willi Gietmann

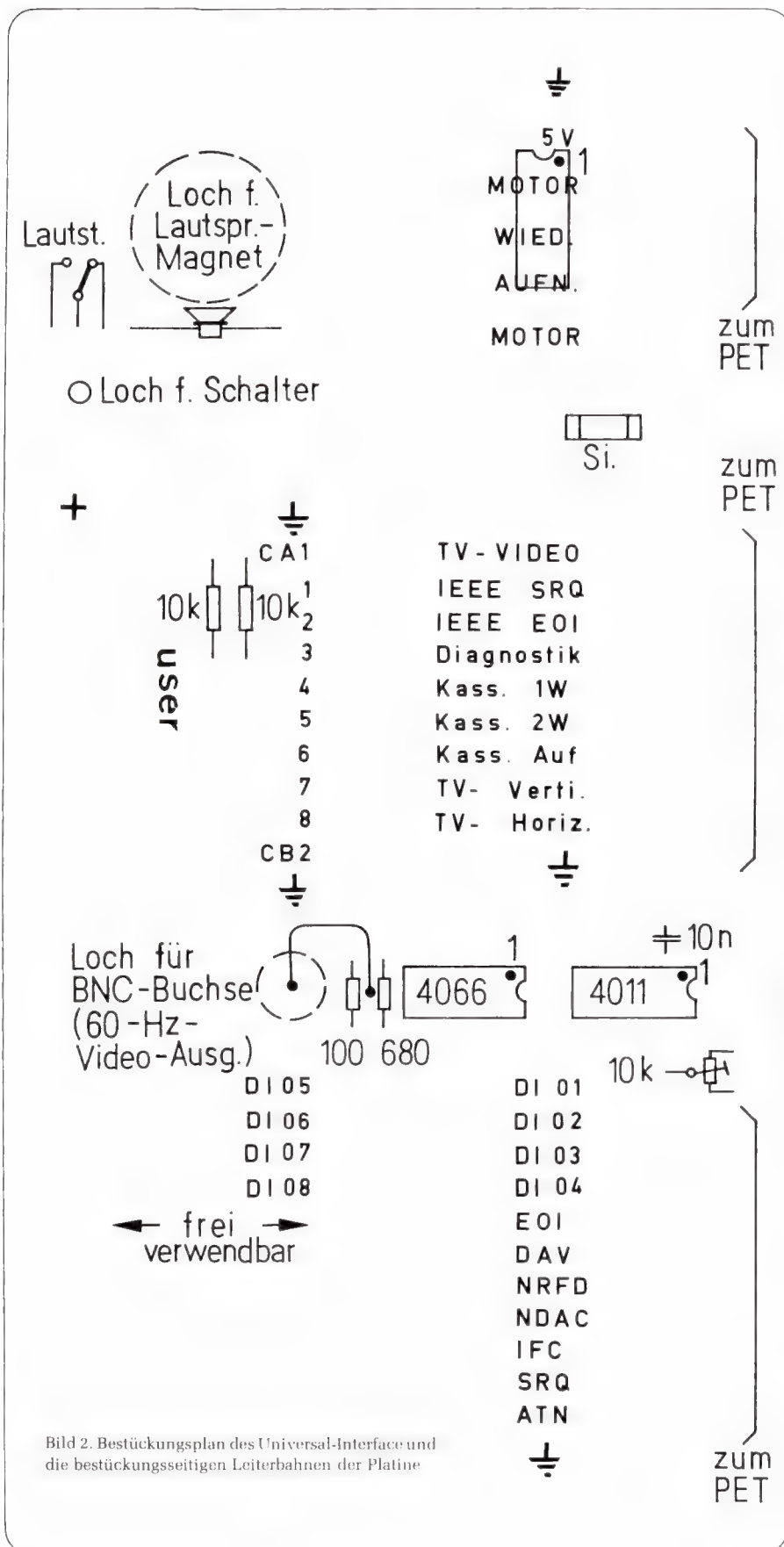


Bild 2. Bestückungsplan des Universal-Interface und die bestückungsseitigen Leiterbahnen der Platine



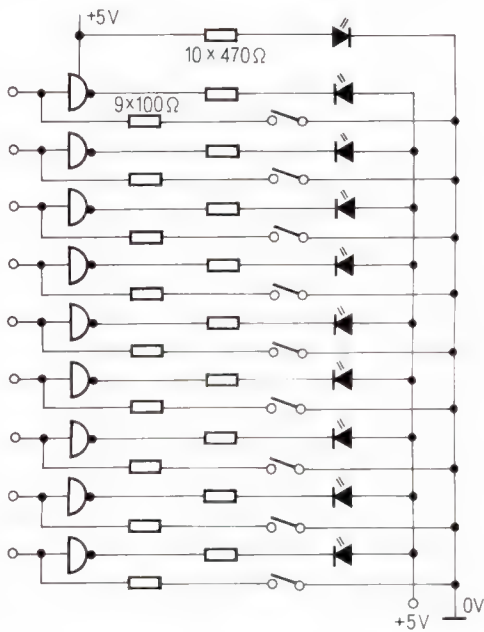


Bild 3. Schaltung zur Simulation von Logikzuständen am I/O-Port

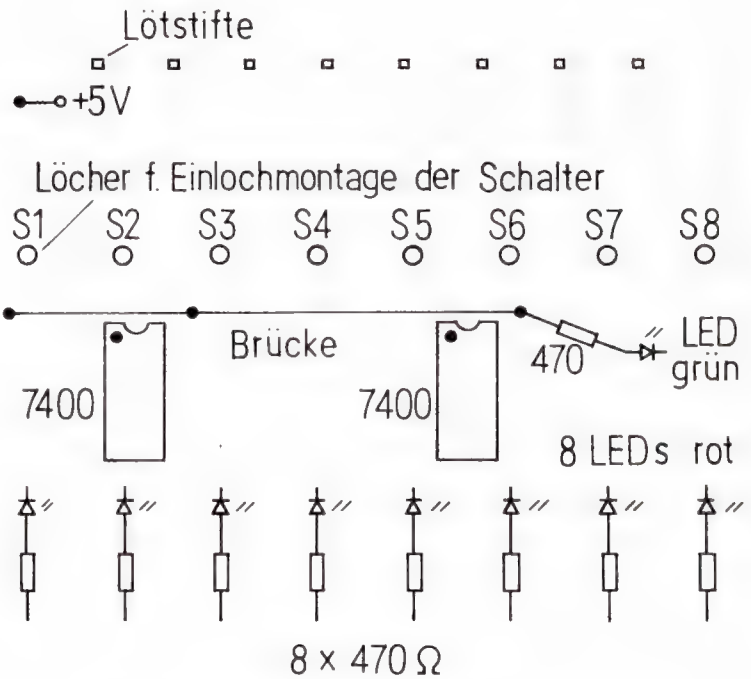


Bild 5. Bestückungsplan der Port-Control-Platine

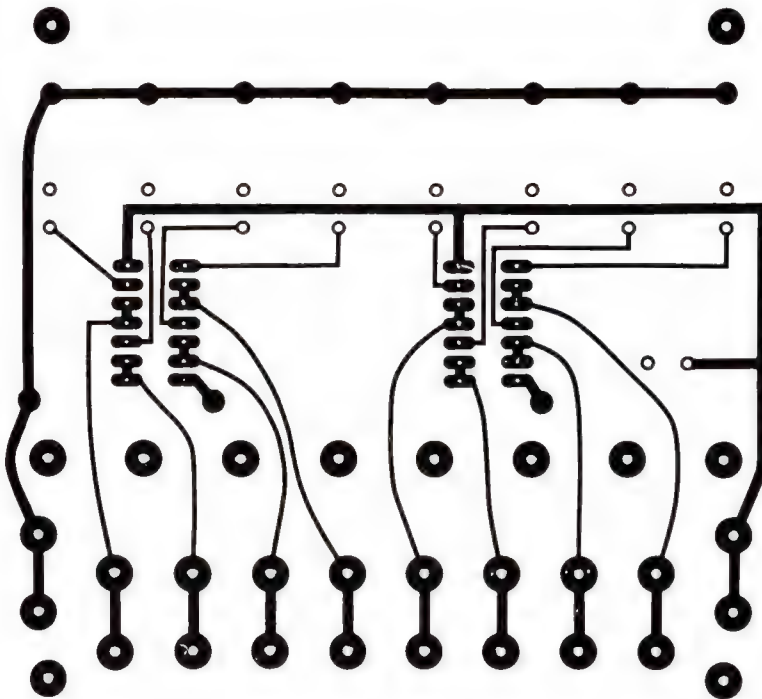


Bild 4. Platinen-Layout der Port-Control-Schaltung

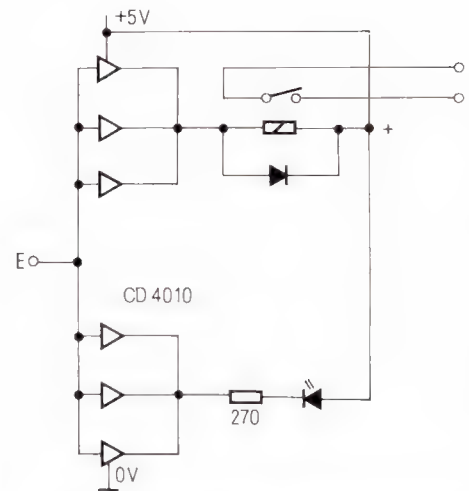


Bild 6. Schaltung eines Telefon-Wählautomaten

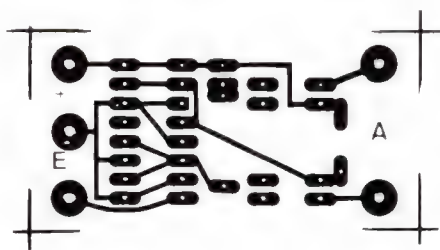
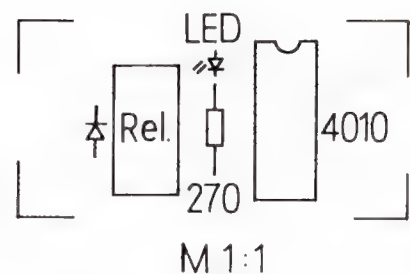


Bild 7. Passende Platine für das Telefon-Interface

Bild 8. Bestückungsplan zu Bild 6 und Bild 7





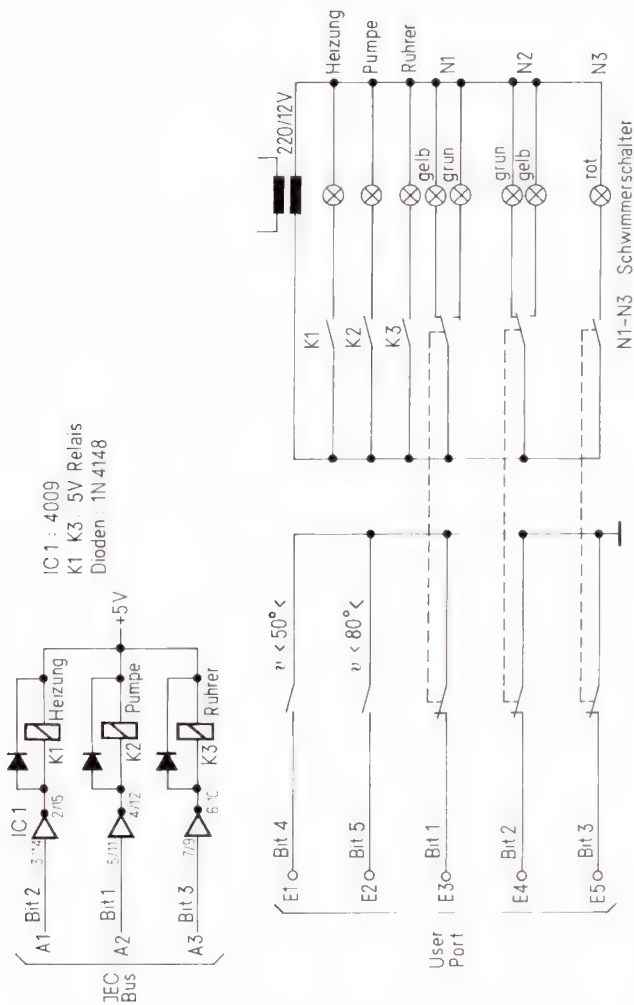


Bild 9. Schaltbeispiel für eine Temperatur- und Wasser-Niveau-Überwachung

```

20 REM ***SCHALTLUHR***
50 DATA HEIZUNG,SPUELMASCHINE,ALARMANLAGE,ROLLREIDEN,
  BELEUCHTUNG,AQUARIUM
60 DATA BEAESSERUNG,KLIMANLAGE
65 UP=59471:POKE59459,255:PW=0:POKEUP,PW
70 FOR I=1T08:READG$(I):PO(I)=80*1+32943:5(I)=PO(I)+20:NEXT
100 INPUT"GWIE SPAET IST ES ";T1$
105 PRINT"AKTUELLE UHRZEIT : ";T1$
110 PRINT"500"
150 PRINT"GERAET 1.2EIT 2.2EIT"
180 PRINT"
190 PRINT"
200 FOR I=1T08
210 PRINTG$(I);TAB(14);"10000-000010000-00001
220 PRINT"
230 NEXT I

```

```

240 PRINT"5000"
250 FOR I=1T08
257 PRINT
260 PRINT"QJNNNNNNNNNNNNNNNNNN";
270 FOR J=1T04
280 FOR K=1T04
290 GETT$:IF T$<>" " THEN 350
300 FOR L=1T0200:NEXT L
310 POKEP0(I),176
320 FOR L=1T0200:NEXT L
330 POKEP0(I),48
340 GOT0290
350 IF ASC(T$)=13 THEN 1000
360 IF ASC(T$)<>32 THEN 600
370 FORM=1T06-K:PRINT"J";:NEXT M
380 PO(I)=PO(I)+6-K
390 GOT0395
600 PRINT$;PO(I)=PO(I)+1
610 T$(I,J)=T$(I,J)+T$
620 IF K<>4 THEN 990
630 PRINT"J";:PO(I)=PO(I)+1
990 NEXT K
995 NEXT J
1000 NEXT I
1004 FOR I=1T08:FOR J=1T04
1006 IF LEN(T$(I,J))<>6 THEN T$(I,J)=T$(I,J)+"0":GOTO1006
1008 T(I,J)=VAL(T$(I,J))
1010 NEXT J,I
1020 TA=VAL(T1$)
1025 PRINT"5NNNNNNNNNNNNNNNNNNNN";:T1$
1030 FOR I=1T08
1040 FOR J=1T04
1050 IF T(I,J)=TA THEN GOSUB1100
1060 NEXT J,I
1070 GOT01020
1100 IF J=10R J=3 THEN P1=5:P2=9:P3=14:PW=PW0R2+(I-1):GOTO1120
1110 P1=1:P2=21:P3=19:PW=PWAND(255-2+(I-1))
1120 POKES(I),P1:POKES(I)+1,P2:POKES(I)+2,P3:POKEUP,PW
1500 RETURN

```

Bild 10. Dieses Programm macht aus dem PET eine denkbar komfortable Schaltuhr

[illegible]

```

180 FORJ=1TOL
185 B$=STR$(A(I)):PRINTB$;
187 IFA(I)=0THENA(I)=10
190 FORJ=1T0A(I)
200 POKE59464,100:POKE59426,1
210 FORK=1T024:NEXTK
220 POKE59464,80:POKE59426,0
230 FORK=1T010:NEXTK
240 NEXTJ
250 FORK=1T0 550:NEXTK
260 NEXTI
270 POKE59467,0
280 GOT010000
1000 DATA1,NOTRUF,HAUSMEISTER,110,VERLAG
1002 DATA2,FEUERWEHR,,112,BETRIEB
1004 DATA3, HUBER,LAGER,399,GALVANIK
1006 DATA4,MEIER,ABRECHNUNG,255,LOHNBUERO
1008 DATA5,HASE,LAGER,398,GALVANIK
1010 DATA6,HUBER,SEKR.,310,EINKAUF
1012 DATA7,DR.RUNOW,ARZT,222,,
2000 DATAENDE
10000 POKE59409,52
10005 PRINT"NOCHNACH MOECHTEN SIE DIE TELEFON-":PRINT
10010 PRINT"NUMMER BESTIMMEN 2000"
10020 PRINT" 1....NUMMERIERUNG"
10030 PRINT"Q 2....NAME"
10040 PRINT"Q 3....KENNZEICHEN"
10042 PRINT"Q 4....GESAMTLISTE"
10044 PRINT"Q 5....HANDRUF"
10046 PRINT"Q 6....KEINE WEITEREN ANRUFEN"
10047 POKE59409,60
10050 GETT$:A=VAL(T$):IFA(<10R)>6THEN10050
10055 IFA=6THENPOKE59467,0:END
10060 ONAGOT010100,10200,10300,10400,100
10070 PRINT"FEHLER! :STOP
10100 PRINT"WOELCHE LAUFENDE NUMMER MOECHTEN SIE ";
10110 INPUTN
10120 GOSUB20000
10130 A$=C$(5*N-1)
10140 GOT0135
10200 PRINT"000"
10210 INPUT"WEN MOECHTEN SIE ANRUFEN ";F$

```



```

10220 FOR I=210255STEP5
10225 IFF$=C$(1) THEN 10250
10230 NEXT I
10235 PRINT "QDER NAME J";F$;" IST NICHT IN":PRINT
10236 PRINT "DER TELEFONDATEI ENTHALTEN!":PRINT:PRINT:PRINT
10238 PRINT "BITTE DRUECKEN SIE IRGEND EINE TASTE!"
10240 GET I$: IFF1$="" THEN 10240
10245 GOTO 10000

10250 N=(I+3)/5:GOTO 1010120
10300 PRINT "Q"
10305 PRINT "WELCHES KENNZEICHEN WUENSCHEN SIE":PRINT "QQQJJJJ";:INPUT F$
10310 FOR I=310253STEP5
10320 IFF$=C$(1) THEN 10350
10325 NEXT I
10330 PRINT "QQQDAS KENNZEICHEN J";F$;" IST NICHT":PRINT
10335 PRINT "(MEHR) IN DER TELEFONDATEI ENTHALTEN.":PRINT:PRINT:PRINT
10340 GOTO 10238

10350 N=(I+2)/5:GOSUB 20000
10360 PRINT "QQMOECHTEN SIE DIESE NUMMER ANNAEHLEN?"
10370 GET A$: IFA$<>"J" AND A$<>"N" THEN 10370
10372 IFA$="J" THEN 10120
10375 NEXT I
10400 PRINT "Q":N=0
10410 FOR I=110255STEP5
10412 PP=VAL(C$(1))/10:IFPP=INT(PP) THEN GOSUB 12000
10415 IFC$(1)="ENDE" THEN 11000
10420 PRINT C$(1);TAB(3);C$(I+1);TAB(16);C$(I+3);TAB(28);C$(I+4)
10425 PRINT
10430 NEXT I
11000 PRINT "QENDE DER LISTE!"
11010 GET I$: IFF1$="" THEN 11010
11020 GOTO 10100

12000 GET I$: IFF1$="" THEN 12000
12005 IFF1$="" THEN 10100
12010 PRINT "Q":RETURN
20000 PRINT "QQQQQ"
20010 PRINT "JJJJ LFD. NR. JJJJ";C$(5*N-4):PRINT
20020 PRINT "JJJJ NAME JJJJJJ";C$(5*N-3):PRINT
20030 PRINT "JJJJ KENNZEICHEN JJ";C$(5*N-2):PRINT
20040 PRINT "JJJJ TEL. NR. JJJJ";C$(5*N-1):PRINT
20050 PRINT "JJJJ W0? JJJJJJJJ";C$(5*N):PRINT:PRINT:PRINT
20060 RETURN

```

```

30000 YY=100
30010 POKE 59464,170
30020 FORK=110YY:NEXT K
30030 POKE 59464,0
30040 FORK=110YY/2:NEXT K
30050 POKE 59464,170
30060 FORK=1104+YY:NEXT K
30070 POKE 59464,0
30080 FORK=110YY*10:NEXT K
30090 GET I$: IFF1$="" THEN 30010
30100 RETURN

```

Bild 11. Der „Telefoncomputer“ kann Ihr Telefonregister verwalten

```

0 REM-----QRB-----
10 PRINT "s"
20 PRINT "DIESES PROGRAMM BERECHNET ENTFERNUNGEN":PRINT
30 PRINT "ZWISCHEN 2 AMATEURFUNKSTELLEN.BITTE":PRINT
40 PRINT "GEBEN SIE STANDORTKENNER IN DER UEB-":PRINT
50 PRINT "LICHEN FORM EIN.":PRINT:PRINT:PRINT
55 PRINT "WENN SIE DAS PROGRAMM BEENDEN MOECHTEN,":PRINT
57 PRINT "GEBEN SIE BITTE EIN '*' EIN!":PRINT:PRINT:PRINT:
    PRINT:PRINT
60 PRINT "BITTE DRUECKEN SIE IRGEND EINE TASTE!"
70 GET I$: IFF1$="" THEN 70
80 PRINT "s"
100 PRINT "BITTE GEBEN SIE DEN QTH-KENNER":PRINT
120 PRINT "DES EIGENEN STANDORTES EIN!"
130 PRINT:PRINT
140 GOSUB 500
160 L1=GL:B1=GB:E$=B$
180 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
200 PRINT "BITTE GEBEN SIE DEN QTH-KENNER":PRINT
220 PRINT "DER GEGENSTATION EIN!"
230 PRINT:PRINT
240 GOSUB 500
260 L2=GL:B2=GB:G$=B$
270 PRINT:PRINT:PRINT:PRINT:PRINT
280 CF=COS(ABS(L1-L2)*PI/180)*COS(B1*PI/180)*COS(B2*PI/180)
300 CF=CF+SIN(B1*PI/180)*SIN(B2*PI/180)
320 F=-ATN(CF/SQR(1-CF*CF))+PI/2

```

```

340 DX=F*6368
350 PRINT"_"
360 PRINT"DIE ENTFERNUNG "E$;"-"G$;" BETRAGT":PRINT
380 PRINT"      "INT(DX+.5);"KILOMETER."
400 L=L+INT(DX+.5):PRINT:PRINT:PRINT
410 N=N+1
420 PRINT"SUMME DER KILOMETER      ":"L
425 PRINT
430 PRINT"ANZAHL DER VERBINDUNGEN  ":"N:PRINT
435 PRINT"SCHNITT (= KM/VERB. )    ":"INT(L/N+.5)
440 GOTO180
500 REM-----EINGABE DER DATEN-----
510 B$=""
515 PRINT"      ";
520 FOR I=1 TO 5
540 GETA$(I):IFA$(I)="" THEN 540
560 IFA$(I)="*" THEN 9000
580 PRINTA$(I);
590 B$=B$+A$(I)
600 NEXT I
602 IFA$(1)<"A"ORA$(1)>"2" THEN 1000
604 IFA$(2)<"A"ORA$(2)>"2" THEN 1000
620 A=ASC(A$(1))-64
640 IFA>14 THEN A=A-26
660 B=ASC(A$(2))-64
680 C=ASC(A$(3))-48
700 D=ASC(A$(4))-48
702 IFC<0 OR C>8 THEN 1000
704 IFD<0 OR D>9 THEN 1000
706 IFC=8 AND D<8 THEN 1000
708 IFC=0 AND D=0 THEN 1000
710 IFD=0 THEN D=10:C=C-1
715 IFA$(5)<"A"ORA$(5)>"J"ORA$(5)="I" THEN 1000
720 IFA$(5)="A" THEN E=2:F=3
740 IFA$(5)="B" THEN E=3:F=3
760 IFA$(5)="C" THEN E=3:F=2
780 IFA$(5)="D" THEN E=3:F=1
800 IFA$(5)="E" THEN E=2:F=1
820 IFA$(5)="F" THEN E=1:F=1
840 IFA$(5)="G" THEN E=1:F=2
860 IFA$(5)="H" THEN E=1:F=3
880 IFA$(5)="J" THEN E=2:F=2

```

```

900 GL=(E+3*D)/15+2*A-2.23333333
920 GB=(F-3*C)/24+B+39.85416666
940 RETURN
1000 PRINT"_"
1020 PRINT"EINGABEFehler!":PRINT:PRINT:PRINT
1040 PRINT"UEBERPRUEFEN SIE DIE EIN-":PRINT
1060 PRINT"GEGEBENEN WERTE!":PRINT:PRINT:PRINT:PRINT
1080 PRINT"FALSCHER STANDORTKENNER : "B$
1100 GOTO180
9000 PRINT"_"
9020 PRINT"SUMME DER KILOMETER      ":"L:PRINT:PRINT
9040 PRINT"ANZAHL DER VERBINDUNGEN  ":"N:PRINT:PRINT
9060 PRINT"SCHNITT (= KM PRO VERB.) ":"INT(L/N+.5)
9100 PRINT:PRINT:PRINT:PRINT
10000 END

```

Bild 12. Funkamateure können mit diesem Programm Entfernungen zwischen zwei QTH-Kennern berechnen

```

50 REM**ROTORSTEUERUNG***
90 RA=59426
95 POKER,8
100 DIMA$(41,5)
110 FOR I=0 TO 42
120 FOR J=0 TO 5
130 READA$(I,J)
135 IFA$(I,J)="ENDE" THEN 170
140 NEXT J
150 NEXT I
160 PRINT"FEHLER!":STOP
165 POKER=59409,52
170 PRINT"_"
172 POKER=59409,52
175 PRINT"QQ IN WELCHEM MODUS MOECHTEN SIE ARBEITEN ?"
180 PRINT"QQ JJ RUFLZEICHEN.....1Q"
190 PRINT"JJ NAME.....2Q"
200 PRINT"JJ WOHNRORT.....3Q"
210 PRINT"JJ ENTFERNUNG.....4Q"
220 PRINT"JJ RICHTUNG.....5Q"
230 PRINT"JJ GRADZAHL.....6"
235 PRINT"JJ AUFLISTEN.....7"

```



```

4060 PRINT"QQ";R$;"JIST NICHT IN DER DATEI ENTHALTEN!";
4070 GOTO2148
5000 PRINT"QQWELCHE ENTFERNUNG MOECHTEN SIE ";
5010 INPUT$
5020 J=3
5030 FORI=0TO41
5040 IFA$(I,J)=R$THEN2080
5050 NEXTI
5060 PRINT"QQ";R$;"JIST NICHT IN DER DATEI ENTHALTEN!";
5070 GOTO2148
6000 PRINT"QQWELCHE RICHTUNG MOECHTEN SIE ";
6010 INPUT$
6020 J=4
6030 FORI=0TO41
6040 IFA$(I,J)=R$THEN2080
6050 NEXTI
6060 PRINT"QQ";R$;"JIST NICHT IN DER DATEI ENTHALTEN!";
6070 GOTO2148
7000 PRINT"QQWELCHE GRADZAHL MOECHTEN SIE ";
7010 INPUT$
7020 J=5
7030 FORI=0TO41
7040 IFA$(I,J)=R$THEN2080
7050 NEXTI
7060 PRINT"QQ";R$;"JIST NICHT IN DER DATEI ENTHALTEN!";
7070 GOTO2148
8000 PRINT"QQMOECHTEN SIE EINE GESAMTUEBERSICHT"
8010 PRINT"ODER EINE EINZELLISTE (G/E) ?"
8020 GETT$:IFT$<>"E"ANDIT$<>"G"THEN8020
8030 IFT$="G"THEN9000
8035 POKES9409,52
8040 PRINT"QDIE LISTE KANN MIT DEN TASTEN > Q"
8050 PRINT"UND < VORWERTS UND RUECKWERTS GE-Q"
8060 PRINT"LESEN WERDEN."
8065 POKES9409,60
8070 PRINT"QQBITTE DRUECKEN SIE EINE TASTE!"
8080 GETT$:IFT$=""THEN8080
8100 FORI=0TO41
8105 POKES9409,52
8110 PRINT"QQQQQ IIRUFZEICHEN...";A$(I,0)
8120 PRINT"QJJNAME.....";A$(I,1)
8130 PRINT"QJJWOHNORT.....";A$(I,2)

```

```

237 POKE59409,60
240 GETI$:A=VAL(I$):IFA=ORA>7THEN240
250 ONAGOT02000,3000,4000,5000,6000,7000,8000
260 STOP
1000 DATA DL0VF,FUNKSCHAU,MUENCHEN,156,OST,90
1999 DATARENDE
2000 PRINT"QQWELCHES RUFZEICHEN MOECHTEN SIE ";
2010 INPUT$
2020 J=0
2030 FORI=0T041
2040 IFA$(I,J)=R$THEN2080
2050 NEXTI
2060 PRINT"QQ",R$;" IST NICHT IN DER DATEI ENTHALTEN!";
2070 GOT02140
2080 PRINT"Q"
2082 POKE59409,52
2083 PRINT"QQJJJJJJ DIE ANTENNE DREHT SICH !"
2086 PRINT"QQJJJRUFZEICHEN...";A$(1,0)
2090 PRINT"QJJNAME.....";A$(1,1)
2100 PRINT"QJJWOHNORT.....";A$(1,2)
2110 PRINT"QJJENTFERNUNG...";A$(1,3)
2120 PRINT"QJJRICHTUNG.....";A$(1,4)
2130 PRINT"QJJGRADZAHL.....";A$(1,5)
2135 POKE59409,60
2137 S=INT(VAL(A$(1,5))/22.5+.5)
2138 GOSUB10000
2140 GETI$:IFI$=""THENGOT02140
2150 GOT0170
3000 PRINT"QQWELCHEN NAMEN MOECHTEN SIE ";
3010 INPUT$
3020 J=1
3030 FORI=0T041
3040 IFA$(I,J)=R$THEN2080
3050 NEXTI
3060 PRINT"QQ",R$;" IST NICHT IN DER DATEI ENTHALTEN!";
3070 GOT02140
4000 PRINT"QQWELCHEN ORT MOECHTEN SIE ";
4010 INPUT$
4020 J=2
4030 FORI=0T041
4040 IFA$(I,J)=R$THEN2080
4050 NEXTI

```

```

8140 PRINT"QJJJENTFERNUNG...";A$(1,3)
8150 PRINT"QJJRICHTUNG.....";A$(1,4)
8160 PRINT"QJJJGRADZAHL.....";A$(1,5)
8165 POKES9409,60
8170 GETT$:IFT$<"<ANDT$<>">"THEN8170
8180 IFT$="<"THEN8300
8190 IFR$(1+1,1)=" "THEN8210
8200 NEXTI
8210 PRINT"QQJJJENDE DER EINZELLISTE !"
8220 GETT$:IFT$=" "THEN8220
8225 IFT$="<"THEN8300
8230 GOT0170
8300 I=1-2
8310 IFI>=0THENNEXTI
8320 PRINT"QQ ANFANG DER LISTE !":I=-1
8330 GETT$:IFT$=" "THEN8330
8335 IFT$=">"THENNEXTI
8340 GOT0170
9000 PRINT"☹"
9005 POKES9409,52
9010 PRINT"CALL NAME WOHNORT QRB RICHTUNG"
9020 PRINT
9030 FORI=0T013
9035 IFR$(1,0)="ENDE"THEN9300
9040 PRINTA$(1,0);TAB(7);A$(1,1);TAB(17);A$(1,2);TAB(27);A$(1,3);
9050 PRINTTAB(31);A$(1,4);TAB(36);A$(1,5)
9060 NEXTI
9065 POKES9409,60
9070 GETT$:IFT$=" "THEN9070
9080 PRINT"☹"
9085 POKES9409,52
9090 PRINT"CALL NAME WOHNORT QRB RICHTUNG"
9100 PRINT
9110 FORI=14T025
9115 IFR$(1,0)="ENDE"THEN9300
9120 PRINTA$(1,0);TAB(7);A$(1,1);TAB(17);A$(1,2);TAB(27);A$(1,3);
9130 PRINTTAB(31);A$(1,4);TAB(36);A$(1,5)
9140 NEXTI
9145 POKES9409,60
9150 GETT$:IFT$=" "THEN9150
9160 PRINT"☹"
9165 POKES9409,52

```

```

9170 PRINT"CALL NAME WOHNORT QRB RICHTUNG"
9180 PRINT
9190 FORI=26T037
9195 IFR$(1,0)="ENDE"THEN9300
9200 PRINTA$(1,0);TAB(7);A$(1,1);TAB(17);A$(1,2);TAB(27);A$(1,3);
9210 PRINTTAB(31);A$(1,4);TAB(36);A$(1,5)
9220 NEXTI
9225 POKES9409,60
9250 GETT$:IFT$=" "THEN9250
9260 GOT0170
9300 POKES9409,60
9305 GETT$:IFT$=" "THEN9300
9310 GOT0170
10000 POKES9467,16:POKE59466,15:POKE59464,100:M=PEEK(RR)
10010 IFM=5THEN10100
10020 IFM>5THEN10060
10021 POKES9426,2
10030 M=M+1:POKEAR,M
10040 FORK=1T0200:NEXTK
10050 GOT010010
10060 M=M-2:GOT010030
10100 PRINT"SQQ"
10105 PRINT"DIE ANTENNE STEHT IN RICHTUNG";M*22.5;
      "GRAD." :POKE59467,0:RETURN

```

Bild 13. Der Computer weiß genau, wohin er den Antennenrotor für bestimmte Gegenstationen drehen muß

```

2 REM**ROULETTE***
4 POKES9467,16:POKE59466,15:FR=59464:POKEFR,0
5 PRINT"☹"
6 Q=RND(1)*255+1
10 IO=59426
20 GETT$:IFT$=" "THENGOSUB1000:GOT020
30 FORI=0T0100STEP4
40 FORJ=1T01:NEXTJ:GOSUB1000:NEXTI
50 PRINT"QQQQQQQQQQQQJJJIDIE ZAHL";PEEK(10);"HAT GEWONNEN!"
60 GETT$:IFT$=" "THEN60
70 RUN
1000 W=W-1:IFW<0THENW=W+16
1005 POKEFR,Q:FORI=1T05:NEXTI:POKEFR,0
1010 POKEIO,W:RETURN

```

Bild 14. Zusammen mit 16 LEDs und einem an den IEC-Bus angeschlossenen Binärdecoder erlaubt dieses kurze Programm das Roulettespiel



```

10 REM**TEMPERATURMESSUNG***
35 PRINT:PRINT
100 Z=PEEK(59471):IF Z=21 THEN 100
110 Z=2:Y=Z1
120 GOSUB 80000
130 GOSUB 80000
140 IF Z=0 THEN 100
145 PRINT"sqjjjjjj QTEMPERATURREGELUNG"
146 PRINT:PRINT:PRINT
150 PRINT"!", " * IN ORDNUNG! * "
155 PRINT
160 IFA(1)=1 THEN 200
161 PRINT"sqjjjjjj QTEMPERATURREGELUNG"
162 PRINT:PRINT:PRINT
163 PRINT"!", " z WASSERNIVEAU ZU TIEF z"
164 PRINT
166 PRINT"          P=EIN H=AUS R=AUS"
170 S1=1:52=1:GOSUB 50000
180 S1=2:52=0:GOSUB 50000
190 S1=3:52=0:GOSUB 50000
195 GOTO 100
200 S1=3:52=1:GOSUB 50000
210 IFA(2)=1 THEN 300
220 S1=1:52=1:GOSUB 50000
230 IFA(4)=1 THEN 260
240 S1=2:52=1:GOSUB 50000
250 GOTO 100
260 S1=2:52=0:GOSUB 50000
270 IFA(5)=0 THEN 100
273 PRINT"sqjjjjjj QTEMPERATURREGELUNG"
274 PRINT:PRINT:PRINT
280 PRINT"!", " % TEMPERATUR ZU HOCH! %"
281 PRINT:PRINT"          NOTABSCHALTUNG DER HEIZUNG"
290 GOTO 100
300 S1=1:52=0:GOSUB 50000
310 IFA(3)=0 THEN 230
312 PRINT"sqjjjjjj QTEMPERATURREGELUNG"
314 PRINT:PRINT:PRINT
320 PRINT"!", " # WASSERNIVEAU ZU HOCH! #"
321 PRINT
322 PRINT"jjjjjjjj P=AUS H=AUS R R=AUS"
39169 :PRINT:PRINT:PRINT" UM ";T$;" ERFOLGTE NOTABSCHALTUNG"

324 PRINTTAB(21)"-----"
327 PRINT:PRINT:PRINT:PRINT:PRINTTAB(10)"BITTE 99 ANRUFEN"
330 GOSUB 61000
335 REM:TONFOLGE
337 POKE 59467,16
340 POKE 59466,15
345 POKE 59464,0
350 POKE 59464,120
355 FOR I=1 TO 100:NEXT I
360 POKE 59464,0
365 FOR I=1 TO 100:NEXT I
370 GETT$:IFT$="" THEN 350
375 POKE 59467,0
380 GOTO 100
40000 Y=Y-224
40030 FOR I=1 TO 5
40040 A(1)=Y AND 2↑(I-1)
40045 IFA(1)>0 THEN A(1)=1
40050 NEXT I
40060 RETURN
50000 S=7-PEEK(59426)
50005 IFS1=3 THEN S1=4
50010 IFS2=0 THEN S0100
50040 S=5 OR S1
50050 POKE 59426,7-S
50055 RETURN
50080 IFS1=0 THEN 50000
50100 S1=7-S1
50110 S=S AND S1
50120 POKE 59426,7-S
50130 RETURN
60000 IFA(4)=0 AND A(5)=1 THEN 60050
60010 IFA(1)=0 AND A(2)=0 AND A(3)=0 THEN Z=1:RETURN
60020 IFA(1)=0 THEN 60150
60030 IFA(1)=1 AND A(2)=0 AND A(3)=1 THEN 60150
60040 Z=1:RETURN
60050 PRINT"sqjjjjjj QTEMPERATURREGELUNG"
60051 PRINT:PRINT:PRINT
60100 Z=0:PRINT"!", " * TEMPERATURFUEHLER DEFEKT! * "
60105 GOSUB 61000
60110 RETURN
60150 PRINT"sqjjjjjj QTEMPERATURREGELUNG"

```

```

60151 PRINT:PRINT:PRINT:PRINT
60200 Z=0:PRINT I$;" * NIVEAUFUEHLER DEFEKT! * "
60205 GOSUB 61000
60210 RETURN
61000 S1=1:S2=0:GOSUB 50000
61010 S1=2:S2=0:GOSUB 50000
61020 S1=3:S2=0:GOSUB 50000
61030 RETURN

```

Bild 15. Temperatur und Wasserniveau überprüft dieses Programm

```

990 REM***PORT-CONTROL***
1000 PRINT"♥
1001 IO=59426:II=IO-2:UI=59471
1002 POKE IO,255
1004 PRINT" BITSTELLE
1005 PRINT"
1010 PRINT" 1 2 3 4 5 6 7 8
1020 PRINT"
1025 PRINT"Q
1030 PRINT"
1040 PRINT" 1 2 3 4 5 6 7 8
1050 PRINT"
1055 PRINT"Q
1060 PRINT"
1070 PRINT" 1 2 3 4 5 6 7 8
1080 PRINT"
1100 FOR I=1 TO 8:IO(I)=32968+3*I:II(I)=IO(I)+200:UI(I)=
IO(I)+400:P%(I)=48+I
1120 N%(I)=176+I:NEXT I
1130 PRINT"QQQ WELCHE BITSTELLE VOM IEC-BUS (1...8)Q"
1140 PRINT" MOECHTEN SIE AENDERN ?"
1150 GET B%:IF B%=0 THEN 1610
1500 H%=PEEK(IO):B%=B%-1
1510 H%=H%+2*B%:W%=P%(B%+1)
1520 IF PEEK(IO(B%+1))<100 THEN H%=H%-2*(B%+1):W%=N%(B%+1)
1530 POKE IO,H%:POKE IO(B%+1),W%
1599 PRINT"S***";PEEK(IO);"I***"
1610 NI%=PEEK(II):IF NI%=RI% THEN 1660
1620 RI%=NI%:FOR I=0 TO 7:IF RI%AND 2*I THEN POKE II(I+1),
P%(I+1):GOTO 1690
1630 POKE II(I+1),N%(I+1)
1640 NEXT I
1650 GOTO 1610
1660 NU%=PEEK(UI):IF NU%=AU% THEN 1710
1670 AU%=NU%:FOR I=0 TO 7:IF AU%AND 2*I THEN POKE UI(I+1),P%(I+1):
GOTO 1690
1680 POKE UI(I+1),N%(I+1)
1690 NEXT I
1700 GOTO 1660
1710 GOTO 1150

```

Bild 16. Auf eine recht übersichtliche Art stellt das Port-Control-Programm den Zustand der PET-I/O-Ports auf dem Bildschirm dar und ist daher eine wesentliche Hilfe bei der Hardware-Entwicklung

## So laufen KIM-Programme auf dem AIM-65

Der Mikrocomputer AIM-65 – er ist identisch mit dem PC-100 von Siemens – verarbeitet sich recht schnell. Kein Wunder – die ASCII-Tastatur, das alphanumerische Display, der Thermo-drucker, das 8-KByte-Monitor-Pro-gramm, die freien PROM-Sockel, das User-VIA 6522 – all das sind gute Ar-gumente für den AIM-65, vom Preis ganz zu schweigen.

Leider gibt es für den AIM-65 noch nicht ganz so viele Programme wie für den KIM-1, und das Umschreiben ist nicht immer leicht. Beim Austauschen der Monitor-Unterprogramme gibt es am wenigsten Schwierigkeiten; die AIM-Monitorroutinen „retten“ meist mehr Register als die entsprechenden des KIM-1. Auch das Umschreiben der I/O-Adressen ist kein Problem, und wenn man am Ende eines Programms beim KIM zur Adresse 1C4F springt, empfiehlt sich beim AIM die Adresse E182.

Probleme ergeben sich beim Um-schreiben aller Programme, die den KIM-Timer (IC 6530) verwenden; der VIA-Baustein 6522 im AIM ist leider völlig anders aufgebaut und braucht mehr Befehle, um angesprochen zu werden. Eine Alternative ist die Ver-wendung des Monitor-Verzögerungs-Unterprogramms „DE2“ bei EC1B, das um (Akkuinhalt mal 256) Mikrosekun-den verzögert. Bei Zeiten von einer Zen-tel Sekunde und mehr, die beim KIM noch möglich sind, beim AIM-6522 aber nicht mehr, schreibt man sich am besten ein kleines Unterprogramm, um im Hauptprogramm nicht durch Ein-schieben zusätzlicher Befehle alle Adressen ändern zu müssen.

Apropos Adressen: Beim KIM-1 ist es ohne weiteres möglich, in Page 1 (0100... ca. 01E0) Programme zu schrei-ben. Der AIM-65 belegt den Bereich 9100...016F aber für sein Monitorpro-gramm. Bei der Zero-Page ist es umge-kehrt: Während der KIM-1 den Bereich 00EF...00FF für das Monitorprogramm belegt, ist dieser Bereich beim AIM-65 frei verwendbar, solange man nicht den Editor in Betrieb nimmt und damit die Adressen ab 00AC belegt. Fe.



# KIM-Klavier

Das in Bild 1 abgedruckte Programm (CPU = 6502) macht aus dem Mikrocomputer KIM-1 ein über die Hexadezimal-Tasten spielbares elektronisches

```

0200 D8          CLD
0201 20 40 1F JSR 1F40
0204 20 6A 1F JSR 1F6A
0207 C9 15      CMP #15
0209 F0 F5      BEQ 0200
020B AA        TAX
020C BD 22 02 LDA 0222,X
020F 8D 05 17 STA 1705
0212 2C 07 17 BIT 1707
0215 10 FB      BPL 0212
0217 A9 01      LDA #01
0219 8D 01 17 STA 1701
021C EE 00 17 INC 1700
021F 4C 00 02 JMP 0200

0222 D5 B9 9E 95 80 6C
0228 5E 55 47 3A 34 29
022E 1F 16 12 0A
    
```

Bild 1. Programmlisting für das KIM-Klavier. Auf die Erzeugung der Halbton-Zwischenschritte wurde hier verzichtet, so daß sich der KIM-1 hier nur in C-dur spielen läßt. Durch Änderung der Notentabelle ist allerdings die Erzeugung der fehlenden Halbtöne unter Verzicht auf den großen Tonumfang möglich

Klavier. Er spielt daher nicht fest vorprogrammierte Melodien, sondern sein Benutzer spielt in „Echtzeitbetrieb“.

Bild 2 zeigt die Belegung der KIM-Tastatur mit den Noten-Namen, und aus Bild 3 geht der Anschluß eines Lautsprechers hervor. Das Programm funktioniert folgendermaßen: Zunächst werden die KIM-Tasten abgefragt. Ist keine Taste gedrückt, so entsteht kein Ton (Akkuinhalt = 15). Wenn dagegen eine Taste gedrückt wird, so wird der Akkuinhalt in das X-Register geladen.

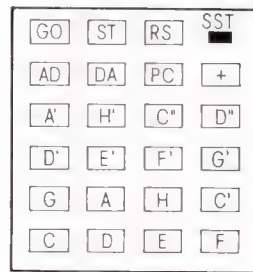


Bild 2. Die Hexadezimal-Tasten des KIM dienen als Klaviatur. Im Prinzip wäre es übrigens auch möglich, noch die Tasten AD, DA, PC, + und GO auszunutzen (Tastencode 10...14)

Der KIM-Timer übernimmt dann einen Hex-Wert aus der Tonfrequenz-Tabelle (0222 + X), der die Periodendauer des erzeugten Tones bestimmt. Sobald die Timer-Zeit abgelaufen ist, wird PA 0 als Ausgang geschaltet und inkrementiert, d.h. hier komplementiert. Dann wiederholt sich das Programm durch einen Sprungbefehl.

Der Tonumfang der Tabelle ist etwas größer als zwei Oktaven, nämlich von C...D“. Die Töne werden mit einem Tastverhältnis von 0,5 erzeugt, so daß sie nur ungeradzahlige Harmonische enthalten und angenehm klingen.

Peter Engels

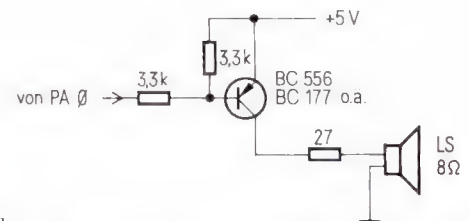


Bild 3. Mit einem PNP-Transistor läßt sich ein kleiner Lautsprecher ansteuern, um die erzeugten Töne hörbar zu machen

## Programme vom Bildschirm fotografiert

Eine der preiswertesten Methoden, Programme und das, was sie auf dem Computer-Bildschirm produzieren, zu verewigen, ist das Fotografieren. Leider ist das nicht ganz so problemlos wie eine Landschaftsaufnahme bei leichter Bewölkung, aber doch problemloser als das Fotografieren bewegter Szenen auf dem Fernsehschirm.

Die Darstellung von Schriftzeichen auf dem Bildschirm eines Computers oder Terminals erfolgt gewöhnlich ebenso wie bei einem handelsüblichen Fernsehgerät nach dem Halbbildverfahren: 50- oder 60mal pro Sekunde schreibt der Elektronenstrahl rund 312 Zeilen auf die Leuchtschicht, bzw. 625 Zeilen in  $\frac{1}{25}$  bzw.  $\frac{1}{30}$  Sekunde. Bei

Belichtungszeiten, die in der Größenordnung von  $\frac{1}{30}$  Sekunden liegen, ist es unvermeidlich, daß Querstreifen auf dem Bild sichtbar werden, da der Elektronenstrahl nicht das gesamte Bild gleichmäßig ausschreibt, während der Kameraschluß offen ist. Form und Größe der Querstreifen hängen dabei von der Art des Verschlusses ab.

Die einfachste Lösung des Problems ist es, Belichtungszeiten von einer halben oder ganzen Sekunde zu wählen. Dazu braucht man allerdings nicht nur ein Stativ, sondern auch ein zitterfreies Bild auf dem Bildschirm. Nun muß man sich nur noch davor hüten, „richtig“ zu belichten; Belichtungsmesser mitteln die Helligkeit über eine bestimmte Flä-

che und lassen außer Acht, daß man bei der Darstellung von Schriftzeichen mit recht konzentrierten weißen Flächen zu tun hat. Um ein „Überstrahlen“ der Schrift zu vermeiden, ist es sinnvoll, die Blende etwas weiter zu schließen, als der Belichtungsmesser empfiehlt.

Als ein in der Praxis erprobter Richtwert kann bei einer Filmempfindlichkeit von 22 DIN und einer Belichtungszeit von einer halben Sekunde eine Blendeneinstellung um 5,6 angesehen werden. Und noch ein Tip: Bildschirme sind nicht eben! Um Kissenverzerrungen zu vermeiden, gehen Sie mit der Kamera nicht zu nahe an den Schirm heran; ein Abstand von mindestens einem Meter ist bei 31 cm Bilddiagonale empfehlenswert.

Fe.

Wolfgang Stanglmeier

# 8080-Disassembler

Bei der Kontrolle von Maschinenprogrammen ist es besonders für den Anfänger sehr mühsam, die hexadezimal oder oktal dargestellten Maschinenbefehle übersetzen zu müssen. Diese Arbeit übernimmt der Disassembler.

Er übersetzt die Maschinenbefehle in den leicht erlernbaren Assemblercode (dieser ist im Elektronik-Sonderheft II, Mikroprozessoren-Software, dargestellt) und gibt diesen auf einem ASCII-Ausgabegerät (Datensichtgerät, Fernschreiber, Drucker) aus.

Der größte Teil des Disassemblers (255 Bytes) besteht aus einer Tabelle, in der alle Maschinenbefehle und Assemblercodes erfaßt sind. Die Tabelle besteht aus mehreren Abschnitten; jeder Abschnitt besteht aus zwei Maskierungsbytes und dem Assemblercode. Die beiden Maskierungsbytes geben an, welche Maschinenbefehle mit dem entsprechenden Assemblercode übersetzt werden.

Jeder Maschinenbefehl wird mit dem ersten Byte Exklusiv-Oder-, mit dem zweiten Byte Und-verknüpft. Wenn das Ergebnis Null ist, dann wird der Maschinenbefehl mit dem folgenden Assemblercode übersetzt, anderenfalls wird die Suche nach dem richtigen Assemblercode im nächsten Abschnitt fortgesetzt.

Bei der Ausgabe der Assemblercodes werden alle Bytes, die größer sind als 1F, direkt ausgegeben; die anderen Bytes führen zum Aufruf von verschiedenen Unterprogrammen, die in Abhängigkeit von verschiedenen Bits im Maschinenbefehl aus anderen Tabellen Abschnitte herausuchen und ausgeben oder 1 oder 2 Bytes Daten ausgeben.

Sobald ein Befehl vollständig ausgedruckt worden ist, wird CR und LF ausgegeben und der Zeiger, der auf den nächsten zu übersetzenden Befehl zeigt, entsprechend erhöht. Wenn die Endadresse noch nicht erreicht ist, wird

der nächste Befehl übersetzt. Anderenfalls wird der Befehl auf Adresse 6097 ausgeführt. Dies kann je nach Anwendersystem ein HALT, ein RST oder ein RET Befehl sein. Bei dem vom Verfasser benutzten System führt der RST-7-Befehl zum Rücksprung in den Monitor.

Da der Disassembler unabhängig von Monitorunterprogrammen ist, dürfte seine Implementierung auf keinem 8080-System ein Problem darstellen. Voraussetzung ist natürlich ein ASCII-Ausgabegerät. An zwei Stellen müssen vom Anwender Daten im Listing an das verwendete System angepaßt werden. An der Adresse 600A muß eingesetzt werden, wieviele Leerzeichen zwischen der Adresse und dem Assemblercode ausgegeben werden sollen. An den Adressen 60C4...60C7 ist Platz für einen CALL-Befehl zu einer eigenen Ausgaberroutine. Das auszugebende Zeichen steht im ASCII-Code im A-Register. BC, DE, HL-Register dürfen verwendet werden. In dem ausgedruckten Disassembler steht an dieser Stelle ein CALL nach 6257; denn auf der Speicherstelle 6257 beginnt eine Ansteuerroutine für einen Nadeldrucker.

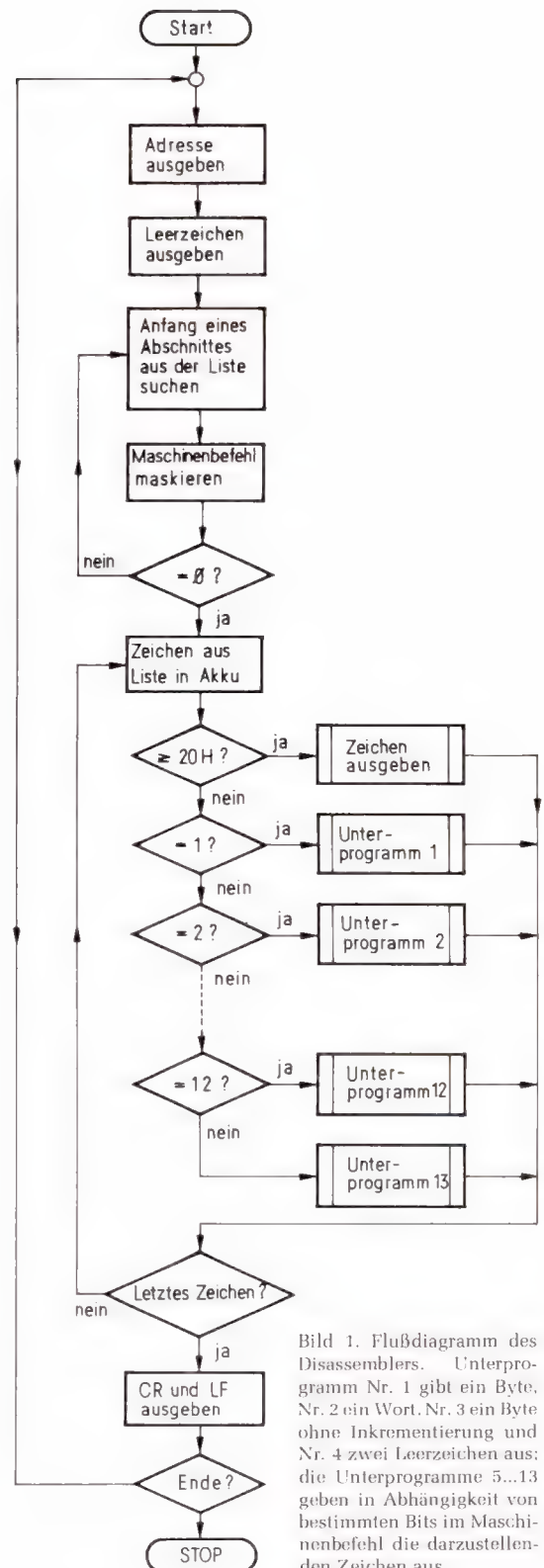


Bild 1. Flußdiagramm des Disassemblers. Unterprogramm Nr. 1 gibt ein Byte, Nr. 2 ein Wort, Nr. 3 ein Byte ohne Inkrementierung und Nr. 4 zwei Leerzeichen aus; die Unterprogramme 5...13 geben in Abhängigkeit von bestimmten Bits im Maschinenbefehl die darzustellenden Zeichen aus



```

6000  E5 78 CD B0 60 79 CD B0 60 2E 04 3E 20 CD C1 60
6010  2D C2 0B 60 21 57 61 7E 23 B7 F2 17 60 0A AE 23
6020  A6 23 C2 17 60 7E E5 11 78 60 D5 E6 7F FE 20 D2
6030  C1 60 3D CA A2 60 3D CA A7 60 3D CA A3 60 3D CA
6040  98 60 11 CC 60 D5 11 07 03 21 EC 60 3D C8 21 04
6050  61 3D C8 2E 1C 3D C8 2E 2C 3D C8 2E 44 3D C8 16
6060  09 3D C8 11 03 04 2E 4C 3D C8 2E 51 3D C8 D1 0A
6070  0F 0F 0F E6 37 C3 C1 60 E1 7E 23 B7 F2 25 60 3E
6080  0D CD C1 60 3E 0A CD C1 60 D1 D5 03 7A 2F 67 7B
6090  2F 6F 09 D2 01 60 E1 FF 3E 20 CD C1 60 3E 20 C3
60A0  C1 60 03 0A C3 B0 60 03 03 0A CD B0 60 0B 0A 03
60B0  F5 1F 1F 1F 1F CD B9 60 F1 E6 0F 06 90 27 CE 40
60C0  27 E5 D5 C5 00 CD 57 62 C1 D1 E1 C9 0A 1F 15 C2
60D0  CD 60 A3 3C 57 7E 23 B7 F2 D5 60 15 C2 D5 60 7E
60E0  E6 7F CD C1 60 7E 23 B7 F2 DF 60 C9 80 41 44 C4
60F0  41 44 C3 53 55 C2 53 42 C2 41 4E C1 58 52 C1 4F
6100  52 C1 43 4D D0 41 44 C9 41 43 C9 53 55 C9 53 42
6110  C9 41 4E C9 58 52 C9 4F 52 C9 43 50 C9 4E DA 5A
6120  A0 4E C3 43 A0 50 CF 50 C5 50 A0 4D A0 52 4C C3
6130  52 52 C3 52 41 CC 52 41 D2 44 41 C1 43 4D C1 53
6140  54 C3 43 4D C3 C2 C3 C4 C5 C8 CC CD C1 C2 C4 C8
6150  53 D0 C2 C4 C8 50 53 D7 00 FF 4E 4F D0 01 CF 4C
6160  58 49 04 0B 2C 82 03 CF 49 4E 58 04 8B 09 CF 44
6170  41 44 04 8B 0B CF 44 43 58 04 8B 04 C7 49 4E 52
6180  04 89 05 C7 44 43 52 04 89 06 C7 4D 56 49 04 09
6190  2C 81 07 C7 88 02 FF 53 54 41 58 20 C2 12 FF 53
61A0  54 41 58 20 C4 22 FF 53 48 4C 44 20 82 32 FF 53
61B0  54 41 04 82 0A FF 4C 44 41 58 20 C2 1A FF 4C 44
61C0  41 58 20 C4 2A FF 4C 48 4C 44 20 82 3A FF 4C 44
61D0  41 04 82 76 FF 48 41 D4 40 C0 4D 4F 56 04 09 2C
61E0  8A 80 C0 05 04 8A C0 C7 52 87 C2 C7 4A 07 04 82
61F0  C4 C7 43 07 04 82 C6 C7 06 04 81 C7 C7 52 53 54
6200  04 8D C1 CF 50 4F 50 04 8C C5 CF 50 55 53 48 20
6210  8C C3 FF 4A 4D 50 04 82 D3 FF 4F 55 54 04 81 E3
6220  FF 58 54 48 CC F3 FF 44 C9 C9 FF 52 45 D4 E9 FF
6230  50 43 48 CC F9 FF 53 50 48 CC DB FF 49 4E 20 04
6240  81 EB FF 58 43 48 C7 FB FF 45 C9 CD FF 43 41 4C
6250  4C 20 82 00 00 2A 83 F5 C5 D3 20 06 0F CD 76 62

```

Bild 2. Hex-Dump des Disassemblers. Will man die Adressen in eine andere Page legen, so sind die unterstrichenen Daten entsprechend zu ändern

Soll das Programm nicht ab Speicherstelle 6000 laufen, so beginnt man mit der Eingabe einfach an einer anderen Adresse, die aber in Hexadezimaldarstellung mit zwei Nullen enden muß, und ändert alle unterstrichenen Bytes entsprechend ab. Wenn Sie zum Beispiel ab Adresse 2E00 eingeben, machen Sie aus allen unterstrichenen 60 einfach 2E, aus 61 machen Sie 2F, aus 62 machen Sie 30. Das ist alles.

Das Programm benötigt einen zusammenhängenden Speicherbereich von 599 Bytes, davon 236 für das Programm, 363 für die Tabellen. Dem Disassembler muß natürlich mitgeteilt werden, welchen Speicherbereich er übersetzen soll. Dazu wird die Anfangsadresse des Bereichs in das Registerpaar BC, die Endadresse in das Registerpaar HL geladen; dann startet man den Disassembler an seiner ersten Adresse. Trifft der Disassembler auf einen nicht erlaubten Befehl, so gibt er ein Sternchen und den betreffenden Befehl im Maschinencode aus.

```

6000  PUSH  H
6001  MOV   A,B
6002  CALL  60B0
6005  MOV   A,C
6006  CALL  6080
6009  MVI   L,04
600B  MVI   A,20
600D  CALL  60C1
6010  DCR   L
6011  JNZ   600B
6014  LXI   H,6157
6017  MOV   A,M
6018  INX   H
6019  ORA   A
601A  JP    6017
601D  LDAX  B
601E  XRA   M
601F  INX   H
6020  JNA   M
6021  INX   H
6022  JNZ   6017
6025  MOV   A,M
6026  PUSH  H
6027  LXI   D,6079
602A  PUSH  D
602B  ANI   7F
602D  CPI   20
602F  JNC   60C1
6032  DCR   A
6033  JZ    60A2
6036  DCR   A
6037  JZ    60A7
603A  DCR   A
603B  JZ    60A3
603F  DCR   A
603F  JZ    6098
6042  LXI   D,60CC
6045  PUSH  D
6046  LXI   D,0307
6049  LXI   H,60EC
604C  DCR   A
604D  RZ

```

Bild 3. Der Disassembler hat sich hier zur Demonstration selbst aufgelistet

604E	LXI	H, 6104	6086	CALL	60C1	60B9	ANI	0F
6051	DCR	A	6089	POP	D	60BB	ADI	90
6052	RZ		608A	PUSH	D	60BD	DAA	
6053	MVI	L, 1C	608E	INX	B	60BE	ACI	40
6055	DCR	A	608C	MOV	A, D	60C0	DAA	
6056	RZ		608D	CMA		60C1	PUSH	H
6057	MVI	L, 2C	608E	MOV	H, A	60C2	PUSH	D
6059	DCR	A	608F	MOV	A, E	60C3	PUSH	B
605A	RZ		6090	CMA		60C4	NOP	
605B	MVI	L, 44	6091	MOV	L, A	60C5	CALL	6257
605D	DCR	A	6092	DAD	B	60C8	POP	B
605E	RZ		6093	JNC	6001	60C9	POP	D
605F	MVI	D, 09	6096	POP	H	60CA	POP	H
6061	DCR	A	6097	RST	7	60CB	RET	
6062	RZ		6098	MVI	A, 20	60CC	LDAX	B
6063	LXI	D, 9403	609A	CALL	60C1	60CD	RAR	
6066	MVI	L, 4C	609D	MVI	A, 20	60CE	DCR	D
6068	DCR	A	609F	JMP	60C1	60CF	JNZ	60CD
6069	RZ		60A2	INX	B	60D2	ANA	E
606A	MVI	L, 51	60A3	LDAX	B	60D3	INR	A
606C	DCR	A	60A4	JMP	60B0	60D4	MOV	D, A
606D	RZ		60A7	INX	B	60D5	MOV	A, M
606E	POP	D	60A8	INX	B	60D6	INX	H
606F	LDAX	B	60A9	LDAX	B	60D7	ORA	A
6070	RRC		60AA	CALL	60B0	60D9	JP	60D5
6071	RRC		60AD	DCX	B	60DB	DCR	D
6072	RRC		60AE	LDAX	B	60DC	JNZ	60D5
6073	ANI	37	60AF	INX	B	60DF	MOV	A, M
6075	JMP	60C1	60B0	PUSH	PSW	60E0	ANI	7F
6078	POP	H	60B1	RAR		60E2	CALL	60C1
6079	MOV	A, M	60B2	RAR		60E5	MOV	A, M
607A	INX	H	60B3	RAR		60E6	INX	H
607B	ORA	A	60B4	RAR		60E7	ORA	A
607C	JP	6025	60B5	CALL	60B9	60E8	JP	601F
607F	MVI	A, 0D	60B8	POP	PSW	60EB	RET	
6081	CALL	60C1						
6084	MVI	A, 0A						

Bild 3. Der Disassembler hat sich hier zur Demonstration selbst aufgelistet

## Rechengeschwindigkeit auf dem Prüfstand

Bei der Programmierung zeitkritischer Rechneranwendungen bieten sich häufig verschiedene Wege an, von denen selbst erfahrene Programmierer nicht immer auf Anhieb zu sagen vermögen, welcher der schnellste ist. In solchen Fällen kommt man um Voruntersuchungen nicht herum, in denen die Verarbeitungsgeschwindigkeit für eine bestimmte Befehlsfolge getestet wird. Vergleichsweise einfach wird ein solcher Test, wenn der Mikrocomputer über eine eingebaute (Software- oder Hardware-)Uhr verfügt, wie das beispielsweise für den PET zutrifft.

In einem solchen Fall setzt man an den Anfang der zu prüfenden Befehlsfolge ein Statement, durch das der Wert im aktuellen Zeitregister in eine Variable eingelesen wird, an den Schluß der Testsequenz wird der Zeitregister-Wert in eine zweite Variable eingelesen, so-

dann die Differenz beider Variablen gebildet und diese ausgedruckt. Ein derartiges Programm für den PET kann so aussehen, wobei noch zu bemerken ist, daß dieser Rechner jede sechzigstel Sekunde quartzgenau und interruptgesteuert parallel zur Abarbeitung von Benutzerprogrammen den Inhalt der Variablen TI (Time, „Zeit“) selbsttätig um eins erhöht:

```
10 Z1 = TI
20... (Testsegment – Beginn)
1000... (Testsegment – Ende)
1010 Z2 = TI: PRINT "VERARBEITUNGSDAUER IN SECHZIGSTEL SEKUNDEN:"; Z2-Z1: END
```

Was aber, wenn der Rechner, wie beispielsweise die Grundausführung von TRS-80 und Euro-Apple, über eine solche eingebaute Uhr nicht verfügt? Hier hat sich das Verfahren bewährt, eine externe Stoppuhr zu verwenden: Man

„leicht“ erst einmal mit Hilfe des folgenden Programms.

```
10 PRINT „UHR AB BITTE“
20 FOR N = 1 TO 1000: GOSUB 1000: NEXTN
30 PRINT „UHR STOP BITTE“: END
1000 RETURN
```

Sodann folgt man den Anweisungen auf dem Bildschirm bei der Bedienung der Stoppuhr und notiert sich den erhaltenen Wert; er stellt den Zeitbedarf zur Meßschleifenbearbeitung dar und ist fortan vom Ergebnis jeder weiteren Messung zu subtrahieren. Nach erfolgreicher Eichung wird nun, beginnend mit Zeile 1000, das zu testende Programmsegment eingefügt und durch die Anweisung RETURN abgeschlossen (RETURN in Zeile 1000 entfällt).

Hans-Georg Joepgen



Vor allem eine Eigenschaft macht die Programmierung in Maschinensprache gegenüber z. B. BASIC umständlicher: Auszugebende Texte für den Dialog mit dem Benutzer können normalerweise nicht innerhalb des Programms stehen. Doch läßt sich dieses Problem leicht lösen.

# „PRTSTR“

## Einfache Textausgabe in Assemblerprogrammen

Ein gutes Dialogprogramm sollte so geschrieben sein, daß es auch von Nicht-Fachleuten bedient werden kann. Diese Forderung wird weitgehend erfüllt, wenn der Programmierer Texte vorsieht, die die Eingabedaten erfragen und Ausgabewerte kommentieren.

In höheren Programmiersprachen (z. B. in BASIC) läßt sich dies leicht durch entsprechende String-Ausgabebefehle erreichen. Schwieriger wird es bei Maschinenprogrammen, vor allem, wenn man von Hand assemblieren muß. Hier wünscht sich mancher Programmierer einen einfachen Druckbefehl, den er benutzen kann, ohne auf Zeiger, Sprungweiten oder Tabellenadressen achten zu müssen. Genau dies bietet ihm das Unterprogramm PRTSTR. Es darf beliebig oft aufgerufen werden; die Länge der auszugebenden

Texte ist nicht begrenzt. Darüber hinaus sind alle Sonderzeichen im String erlaubt.

Das Unterprogramm wird nach dem Schema in *Bild 1* aufgerufen. Der Text ist nach dem JSR-Aufruf eingefügt. Er wird durch den Befehl NOP (hex EA) abgeschlossen. \$EA ist somit die einzige Bitkombination, die nicht durch PRTSTR ausgegeben werden kann. Nach der NOP-Anweisung wird das Maschinenprogramm wie gewohnt fortgesetzt.

*Bild 2* zeigt den Ablauf des Unterprogramms PRTSTR. Durch den Unterprogrammsprung bleibt die Adresse des auf das Befehlswort folgenden Byte zurück. Das Programm lädt diese nun in einen Zeiger und druckt mit dessen Hilfe alle folgenden Bytes als ASCII-Zeichen auf dem angeschlossenen Terminal aus. Dazu wird ein Unterpro-

gramm benutzt, das ein im Akku stehendes Zeichen ausgibt. Im KIM-1-System ist dies die Routine OUTCH (Adr. 1EA0). Bei Betrieb auf anderen Rechnern trägt man hier die Adresse einer entsprechenden Routine ein. Die Inhalte der Register und des Akkus dürfen zerstört werden. Das Programm selbst ist „relocatable“ geschrieben, d.h. es kann ohne Änderung an beliebiger Stelle (auch im PROM) im Hauptspeicher abgelegt werden.

Noch ein Hinweis: PRTSTR benutzt zum Rücksprung den Befehl JMP-indirect (6C). Bedingt durch einen Maskenfehler bei zahlreichen 6502-CPU's tritt ein Fehlverhalten auf, wenn der Text zufällig an einer Page-Grenze endet (FUNKSCHAU 1979, Heft 10, Seite 582). Sollte dies der Fall sein, so kann man dies durch Einfügen eines Leerwortes (hex 00) umgehen.

Erich H. Franke

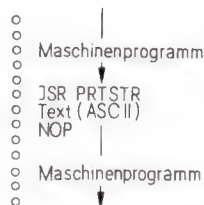


Bild 1. Einfügen des Unterprogramms PRTSTR in ein Maschinenprogramm. Dem Unterprogrammaufruf folgt sofort der auszugebende Text als Folge von ASCII-Zeichen, die mit hex EA abgeschlossen wird

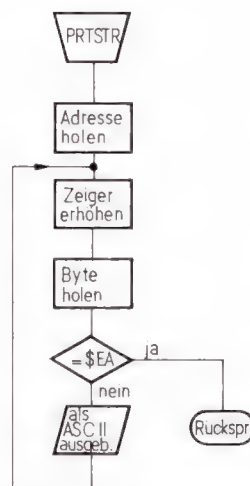


Bild 2. Flußdiagramm zum Ablauf des Textausgabe-Unterprogramms. Der Rücksprung erfolgt indirekt über eine Zero-Page-Adresse

0000	68		PLA
0001	85	F8	STA F8
0003	68		PLA
0004	85	F9	STA F9
0006	E6	F8	INC F8
0008	D0	02	BNE 000C
000A	E6	F9	INC F9
000C	A0	00	LDY #00
000E	B1	F8	LDA (F8),Y
0010	C9	EA	CMP #EA
0012	F0	06	BEQ 001A
0014	20	A0 1E	JSR 1EA0
0017	18		CLC
0018	90	EC	BCC 0006
001A	6C	F8 00	JMP (00F8)

Bild 3. Das Programm, hier für die CPU 6502, ist recht kurz und beliebig verschiebbar

In manchen Situationen hat ein Computer nicht genug Zeit, um zwischen zwei Programmteilen das Ergebnis eines Prozesses auf einem Terminal auszudrucken. Ein typisches Beispiel ist der Empfang von Baudot-Fernschreibsendungen mit 75 Bd und deren Ausdruck auf einem ASCII-Sichtgerät mit 110 Bd; die Zeit zwischen zwei Baudot-Zeichen ist einfach zu kurz. Abhilfe schafft eine Interrupt-Routine, die für eine Ausgabe ohne Beeinträchtigung des Hauptprogrammes sorgt.

# ASCII-Ausgabe per Interrupt

## Die Interrupt-Routine

Bereits in Heft 7/1979 der FUNKSCHAU war ein Programm beschrieben, das es gestattet, innerhalb einer Interrupt-Routine vom Terminal ein ASCII-Zeichen in einen Zwischenspeicher einzulesen, während die eingetippten Zeichen gleichzeitig auf dem Display des Mikrocomputers KIM-1 als Siebensegment-Pseudoalphabet zu sehen sind. Hier wird nun genau das Gegenteil beschrieben: die Ausgabe von ASCII-Zeichen über einen Bufferspeicher (FIFO) per Interrupt-Routine.

Das dafür entwickelte Interrupt-Programm ist für die CPU 6502 und die I/O-Adressen des KIM-1 ausgelegt (Bild 1). Es benutzt den KIM-Interrupt-Timer zur Erzeugung eines periodischen Interrupts; er wird nach dem ersten Anspringen der Interrupt-Routine von dieser selbst immer wieder neu gestartet und verzögert um eine Bitlänge. Die im Listing ausgedruckte Verzögerungszeit ist für 600 Bd ASCII-Geschwindigkeit ausgelegt (hex 1A), kann aber beliebig geändert werden.

Das Programm gibt die in einem FIFO-Buffer (0200...02FF) stehenden Zeichen aus und hält an, sobald es auf das ASCII-Zeichen NUL (hex 00) stößt. Als FIFO-Index dient die Zero-Page-Zelle 00E2; 00E1 ist der Zeichenbuffer, und 00E0 wird als Bitzähler verwendet. (FIFO bedeutet „First In, First Out“.)

## Das Baudot-Empfangsprogramm

Wer einen KIM-1 und ein ASCII-Terminal oder einen ASCII-Fernschreiber besitzt, kann das Interrupt-Programm gleich nutzbringend einsetzen, z. B. um Fernschreibsendungen auf Kurz- oder Langwelle zu empfangen (Vorsicht: postalische Bestimmungen beachten!). Wer dazu berechtigt ist, kann auch Nachrichtenagenturen mit-schreiben; sie senden zum Teil mit 50 Bd, aber auch mit 75 Bd. Bild 2 zeigt die erforderliche Baudot-ASCII-Umwandlungstabelle, und Bild 3 die Programm-

segmente für den Baudot-Empfang, hier für 75 Bd (die Geschwindigkeit läßt sich an den Adressen 0117 und 0128 beliebig ändern; hier müssen die Verzögerungszeiten für etwa eine halbe und eine ganze Baudot-Bitlänge in ms hexadezimal stehen).

Ein Funkfern-schreibkonverter (s. FUNKSCHAU 9/1979) ist am Port PB 3 des KIM anzuschließen. Auf dem wie üblich angeschlossenen ASCII-Termi-

nal kann nun der empfangene Text mit-gelesen werden. Da Terminals meist nur 64 Zeichen pro Zeile darstellen können, die Fernschreibsender aber mit 72 Zeichen breiten Zeilen arbeiten, wird das Format beim Empfang entsprechend geändert. Zu diesem Zweck wertet das Programm aus, ob gegen Ende einer Terminal-Zeile ein Leer-raum vorhanden ist und trennt bei die-sem, um keine Worte zu „zerreißen“.

Bild 1. Interrupt-Routine zur Ausgabe von ASCII-Zeichen aus einem FIFO-Buffer zwischen 0200 und 02FF. Die Adressen der I/O-Ports beziehen sich auf den KIM-1

0069	48	PHA	AKKU UND
006A	98	TYA	Y-REG.
006B	48	PHA	RETTE
006C	A5 E0	LDA E0	BITZAEHLER
006E	30 1E	BMI 008E	TESTEN
0070	C6 E0	DEC E0	U. DEKREM.
0072	C9 02	CMP 02	STOP BIT
0074	B0 03	BCS 0079	BEI BIT-
0076	38	SEC	ZAEHL. < 2
0077	B0 02	BCS 007B	
0079	46 E1	LSR E1	BIT ZUM
007B	AD 42 17	LDA 1742	AUSGANG
007E	29 FE	AND 0FE	SCHIEBEN
0080	69 00	ADC 00	
0082	8D 42 17	STA 1742	
0085	A9 1A	LDA 01A	TIMER
0087	8D 0E 17	STA 170E	STARTEN
008A	68	PLA	
008B	A8	TAY	RUECK-
008C	68	PLA	SPRUNG
008D	40	RTI	
008E	A9 0A	LDA 00A	BITZAEHLER
0090	85 E0	STA E0	LADEN
0092	A4 E2	LDY E2	INDEX
0094	B9 00 02	LDA 0200, Y	CHR. HOLEN
0097	0A	ASL A	
0098	85 E1	STA E1	E1=BUFFER
009A	F0 02	BEQ 009E	00=ENDE
009C	E6 E2	INC E2	INDEX
009E	18	CLC	ERHOEHEN
009F	90 E4	BCC 0085	



Außerdem wird an der empfangenen Zeilenlänge festgestellt, ob ein Absatz gemacht wurde; wenn ja, so wird auch auf dem Terminal ein Absatz eingefügt, um die Übersichtlichkeit des Textes zu wahren. Wird das Trennungszeichen NNNN empfangen, so wird auf das Terminal zusätzlich ein frei programmierbarer Text (01AF...018C) ausgegeben; selbstverständlich geschieht das wieder über den Ausgabe-FIFO, damit kein empfangenes Zeichen verloren geht.

Die Startadresse des Beispielprogramms ist 0000. Zu Beginn werden zunächst die Empfangs- und Ausgabe-FIFO-Indices 0200 und der NMI-Vektor auf 0069 gesetzt (natürlich muß NMI mit dem Timer-Interrupt-Pin PB 7 verbunden sein) und der Timer initialisiert, um den Interrupt das erste Mal auszulösen. Das Unterprogramm zum Baudot-Empfang beginnt bei 0105. Außer den Möglichkeiten der Interrupt-Ausgabe wird dabei auch deutlich, was „Echtzeit-Textverarbeitung“ heißt!

Herwig Feichtinger

#### Stichworte zum Inhalt

6502, KIM-1, Baudot-ASCII, Timer-Interrupt, Funkfern schreiben, RTTY.

#### TABELLE BAUDOT-ASCII

00A1	24	45	00	41	20	53	49	55
00A9	00	44	52	4A	4E	46	43	4B
00B1	54	5A	4C	57	48	59	50	51
00B9	4F	42	47	00	4D	58	56	00
00C1	44	33	00	2D	3E	27	38	37
00C9	00	00	34	3B	2C	5B	3A	28
00D1	35	2B	29	32	21	36	30	31
00D9	39	3F	5D	00	2E	2F	3D	E8

◀ Bild 2. Tabelle zur Codeumwandlung. Das Baudot-Zeichen dient hier als Adressenindex zum Auffinden des in der Tabelle stehenden äquivalenten ASCII-Zeichens

Bild 3. ▼  
Programm zum Empfang von Baudot-Fernschreibsendungen

0000	A9	00	8D	FB	17	8D	00	02	85	F3	85	E2	85	E5	A9	69
0010	8D	FA	17	8D	0F	17	E6	E4	20	0F	01	C9	08	Do	1F	20
0020	0F	01	C9	02	Do	Fo	A6	E4	A9	00	85	E4	E0	3A	30	08
0030	A9	20	20	52	01	4C	16	00	20	5C	01	4C	18	00	4C	6D
0040	01	EA	EA	EA	EA	C9	04	Fo	E7	C9	02	Fo	E8	05	F3	AA
0050	BD	A1	00	Fo	C3	20	52	01	4C	16	00	A4	E5	99	00	02
0060	E6	E5	C8	A9	00	99	00	02	60							
0105	48	A5	E2	8D	0D	17	A9	00	85	F3	A9					
0110	08	2C	02	17	Do	FB	A2	05	8E	47	17	2C	02	17	Fo	F6
0120	2C	47	17	10	F6	A0	05	A2	0D	8E	47	17	2C	47	17	10
0130	FB	18	2C	02	17	Do	01	38	66	Fo	88	Do	EC	A5	Fo	4A
0140	4A	4A	EA	Fo	02	C9	1F	Fo	C2	C9	1B	Do	04	A9	20	Do
0150	BC	60	C9	20	Do	11	A4	E3	Co	30	30	0B	A9	0D	20	5B
0160	00	A9	00	85	E3	A9	0A	20	5B	00	E6	E3	60	C9	0C	Fo
0170	07	A2	04	86	E6	4C	45	00	C6	E6	Do	F9	A2	23	BD	8C
0180	01	20	52	01	CA	10	F7	86	E3	4C	16	00	2A	2A	2A	2A
0190	2A	2A	2A	20	45	43	49	56	52	45	53	20	53	57	45	4E
01A0	20	42	59	31	43	44	20	2A	2A	2A	2A	2A	2A	2A	0A	0D

## Computer-Hobbyisten: Was sind das für Leute?

In Werbeprospekten zu Erzeugnissen für den Hobby-Computer-Markt wird häufig betont, daß zum Steckenpferd-Programmieren keinerlei elektronische Vorkenntnisse erforderlich seien und mit dem betreffenden Produkt das Tor zu einem „Hobby für jedermann“ aufgetan werde. Stimmt das dergestalt entworfene Bild mit der Wirklichkeit überein, ist das Programmieren nach Feierabend eine Beschäftigung, der auch Leute ohne „elektronische Vorgesichte“ anhängen – was sind das für Steckenpferd-Reiter, die Computerfans von heute?

Südwestfunk-Reporter wollten Genaueres über die Zusammensetzung der Gruppe der Mikrocomputer-Fans wissen und starteten eine Befragungsaktion, in der etwas mehr als 100 Computerfreunde aus dem südwestdeutschen Raum mündlich und telefonisch nach ihrer Vorgeschichte befragt wurden: Eine Erhebung, die wegen der unsystematischen Auswahl der Befragten und wegen ihrer vergleichsweise geringen Zahl nicht den Anspruch demoskopischer Genauigkeit erheben will

und der deswegen auch kein Repräsentativcharakter zukommt, die aber sehr wohl geeignet ist. Tendenzen aufzuzeigen.

Danach überwiegt noch immer deutlich die Gruppe der „Profis“ und der „Halbprofis“ bei der in Frage kommenden Käuferschicht für Mikrocomputer-Artikel, sie stellen noch immer mehr als zwei Drittel aller Befragten. Profis – das sind beispielsweise Techniker und Ingenieure aus Elektronik-Berufen oder Disziplinen wie Maschinenbau sowie Luft- und Raumfahrttechnik, die im Rahmen ihrer Ausbildung oder Berufsausübung mit Digitalelektronik in Berührung kamen; zu den Halbprofis zählen in der Mehrzahl Hobbyisten, die als Funkamateure, CB-Fans oder Elektronikbastler erste Bekanntschaft mit dem Ohmschen Gesetz und den Grundlagen der TTL-Technik machten. Beide Gruppen stellen also, wie gesagt, den Löwenanteil. Wie aber setzt sich die verbleibende Gruppe der „Nicht-Elektroniker“ zusammen? Hier dominieren – und dies ist ein erfreuliches Anzeichen für die Aufgeschlossenheit man-

cher Schulen für neue Entwicklungen – vor allem Lehrer naturwissenschaftlicher Disziplinen an beruflichen Schulen, Gymnasien und – in zwei Fällen – Realschulen sowie Schüler, die in von diesen Lehrern eingerichteten Arbeitsgemeinschaften und Computer-Clubs mitwirken.

Den Rest der Befragten stellten ein Handelsvertreter für Damen-Oberbekleidung, zwei Theologie-Studenten, ein Polizeibeamter, fünf kaufmännische Angestellte sowie der Angehörige eines bekannten südwestdeutschen Symphonie-Orchesters, der sich dagegen wehrte, in die Gruppe der „Nicht-Fachleute“ eingereiht zu werden, da er beim Aufbau seiner Modelleisenbahn-Anlage im Keller elektrische Erfahrungen gewonnen habe und seinem KIM in durchaus professioneller Weise zur Eisenbahn-Steuerung einsetze: Wie dem auch sei, das Bild vom Volks-Steckenpferd für Nichtfachleute wird durch das Ergebnis dieser Erhebung vom Frühsommer 1979 noch nicht bestätigt.

Hans-Georg Joepgen



## HOFACKER

Tegernseer Straße 18  
D-8150 Holzkirchen  
Telefon (08024) 73 31

Lieferung durch den  
Fachhandel oder per  
NN- oder Voraus-  
kasse.  
PSchK. München  
1959 94-807  
oder Scheck.  
Empfohlene  
Verkaufs-  
preise.

## HOFACKER-VERLAG

### 6502 USER NOTES

Die Information für  
den 6502 Fan.  
Artikel, Software  
u. viele Schaltungen  
u. Tips für  
KIM, SYM, AIM,  
PET und Ohio.  
Aus dem Inhalt:

Disassembler für KIM, ein Programm zur Erzeugung großer  
Buchstaben auf einem Drucker, Checkout, ein interessantes  
Programm zum Test, Korrigieren und Ändern auf dem KIM.  
Zugriff zum SYM-1 Display u. v. a. mehr.

Drei Hefte zum Kennenlernen, Best.-Nr. 220 a .. **DM 29.—**  
Dr. Dobbs Vol. II, Eine wertvolle Informationsquelle.  
9 Hefte aus 1977 .. **DM 79.—**  
Bestell-Nr. 212  
Dr. Dobbs, Vol. III, 10 Hefte aus 1978 komplett **DM 89.—**  
Bestell-Nr. 213  
What to do, after you hit Return  
Das große erste Spielbuch für BASIC-Programme **DM 29.80**  
Bestell-Nr. 217

### Bewährte Elektronik- Bestseller

Best.- Nr.	Titel	Preis DM
1	Transistor-Berechnungs- und Bauanleitungs- Handbuch, Band 1 .....	19.80
2	TBB, Band 2 (Fortsetzung von Nr. 1) .....	19.80
3	Elektronik im Auto mit Handb. f. Polizei-Radar .....	9.80
4	IC-Handbuch (TTL, CMOS, Linear) .....	19.80
5	IC-Datenbuch, Anschlußbilder und Kurzdaten, TTL, CMOS, Linear .....	9.80
6	IC-Schaltungssammlung .....	9.80
7	Elektronikschaltungen zum Basteln .....	5.—
8	IC-Bauanleitungs-Handbuch .....	19.80
9	Feldeffekttransistoren, Grundl., Schaltbeispiele ..	5.—
10	Elektronik und Radio, IV, erweiterte Auflage, Best.-Nr. 10 .....	19.80
11	IC-NF-Verstärker, Schaltbeisp. und Daten .....	9.80
12	Beispiele integrierter Schaltungen .....	19.80
13	Hobby-Elektronik-Handbuch .....	9.80
14	IC-Vergleichsliste, TTL, CMOS, Linear (neu) ..	29.80
15	Optoelektronik-Handbuch .....	19.80
16	CMOS, Teil 1, Einführung, Schaltbeispiele .....	19.80
17	CMOS, Teil 2, Fortsetzung von Nr. 16 .....	19.80
18	CMOS, Teil 3, Fortsetzung von Nr. 16 und 17 ..	19.80
19	IC-Experimentier-Handbuch .....	19.80
20	Operationsverstärker, Grundlagen und Schalt- beispiele, 175 Seiten .....	19.80
21	Digitaltechnik-Grundkurs, eine Einführung vom Grundgatter zum Microcomputer .....	19.80
23	Elektronik-Grundkurs, die ideale Einführung in die Elektronik vom Elektron bis zum Operations- verstärker. Über 160 Seiten .....	9.80
800	Master Handbook. 1001 praktische elektronische Schaltungen .....	49.—
101	CB-Handbuch f. Hobbyfunk, Tips u. Schaltungen	19.80

### Mikroprozessor-Bücher in deutscher Sprache

22	Mikroprozessoren, Grundlagen, Eigenschaften und Aufbau. Über 140 Seiten. 2., völlig neu über- arbeitete Auflage .....	19.80
26	Mikroprozessoren, Teil 2, Fortsetzung von Nr. 22, Schaltbeispiele usw. Über 140 Seiten (8080) ...	19.80

Best.- Nr.	Titel	Preis DM
25	Hobby-Computer-Handbuch. Die ideale Einführung in die Microcomputer-Technik für Anfänger. Über 350 Seiten .....	29.80
27	Microcomputer-Software-Handbuch. Grundlagen der Programmierung in Maschinensprachen, Assembler, Compiler, Interpreter, BASIC (8080, 6800, Z 80, 6502, 2650, SC/MP u. v. a.), Be- fehlislisten. Über 250 Seiten .....	29.80
33	Microcomputer-Programmierbeispiele. Grundlagen des Systems 2650 mit vielen Programmierbei- spielen (Stoppuhr, Musikprogramm, Zähler, Lauf- licht, Reaktionstest u. v. a.). Zu jedem Programm genaue Beschreibung m. Flußdiagramm u. Listing. Ca. 120 Seiten .....	19.80
24	Microcomputer-Technik von Blomeyer-Bartenstein. Der erste Autor des ersten deutschsprachigen Buches über Microcomputer gibt eine umfas- sende einführende und weiterführende Hilfe zum Einstieg in die Microcomputer-Technik. Ob. 250 S.	29.80
N 8	SC/MP Programmier- und Assembler-Handbuch ..	19.80
80/63	M 6800 Programmierhandbuch .....	19.80
109	6502 Microcomputer Aufbau u. Programmierung	29.80
110	Programmierhandbuch für PET .....	29.80
113	BASIC Programmierhandbuch .....	19.80
28	Microcomputer Lexikon und Wörterbuch .....	29.80

201	32 BASIC Programs for the PET Computer	52.—
202	32 Programme aus 201 auf 5 Cassetten ..	149.—
205	Introduction to TRS-80 Graphics .....	29.80
222	The Best of PET Gazette Viele Programme für PET 2001 8K-Besitzer	39.80
1011	How to Design, Build and Program your own working Computer System .....	29.80
1076	Artificial Intelligence .....	29.80
1099	How to Build your own working 16 Bit Microcomputer (TMS 9900) .....	14.80
1053	Microprocessor Cookbook .....	24.80
150	Care and Feeding of the Commodore PET Hardware Manual .....	19.80
151	Microsoft 8K BASIC Manual .....	19.80
152	Expansion Handbook for 6502 and 6800 ...	19.80
153	Microcomputer Application Notes (Intel) ....	29.80
154	Complex Sound Generation with SN 76477	19.80
155	The first Book of 80-US (Software+Tips+ Programmiertricks für TRS-80) .....	19.80
34	TINY-BASIC Handbuch in deutsch. Einführung, Programmbeispiele, Hardware Tips .....	19.80
80/13	TINY-BASIC auf Cassette mit Manual ....	79.—
31	57 Programme in BASIC (deutsch) .....	39.—
111	Programmieren mit TRS-80, ein Buch nur für den TRS-80. Einführung, Programmiertricks, Zusatzschaltungen und viele Programmier- beispiele (deutsch) .....	29.80
1141	How to built your own Robot PET .....	29.80

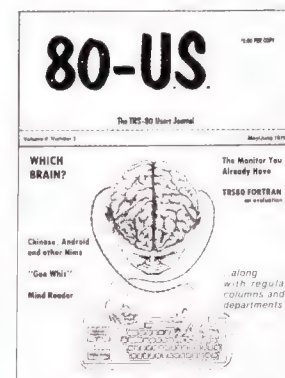
### Datenbücher Allgemein

N 1	TTL-Databook .....	19.80
N 2	Linear Datenbuch .....	24.80
N 23	Linear Applications, Vhume (neu) .....	19.80
N 3	Linear Applications, Volume 2 .....	14.80
N 4	Voltage Regulator Handbook .....	9.80
N 5	Audio Handbook .....	14.80
N 6	Special Function Data Book .....	9.80
N 7	c mos Integrated Circuit Data Book .....	9.80
N 7/1	MOS LSI Data Book .....	24.80
N 8	SC/MP Programmier- u. Ass.-Handbuch .....	19.80
N 9	SC/MP Microprocessor Applications .....	19.80
N 20	Memory Data Book .....	19.80
N 21	Interface Integrated Circuits Data Book ....	19.80
N 22	Data Acquisition Handbook .....	19.80
N 23	Power Transistor Databook .....	9.80
N 24	FET Databook .....	9.80
1010	Transistorvergleichsliste 2 N, A—Z, auch jap.	19.80

### Microprocessor-Bücher in englischer Sprache

1085	24 Ready to RUN Programs in BASIC for PET and TRS-80 .....	24.80
1095	Programs in BASIC for Electronic Engineers	19.89
1088	Illustrated Dictionary of Microcomputers ....	35.80
1071	Complete Handbook of Robotics .....	29.80
785	Microprocessor/Microprogramming Handbook, 290 Seiten .....	35.—
985	Programming Microprocessors, 290 Seiten....	35.—

Best.- Nr.	Titel	Preis DM
752	Computer Programming Handbook. Der „Ein- Buch-Computer-Kurs“. Über 510 Seiten .....	49.—
574	Beginner's Guide to Computer Programming ..	39.—
N 9	SC/MP Microprocessors Application Handbook	19.80
709	Modern Guide to Digital Logic Memories, processors and interface .....	35.—
952	Microprocessors Programming for Computer Hobbyists .....	39.—
80/20	Dr. Dobbs' Journal, Vol. 1 .....	59.—
80/21	Basic Programm Bibliothek, 5 Bücher mit BASIC-Programmen .....	425.—
80/22	2650 Microprocessor Manual .....	29.80
80/6	MCS 85 Users Manual 8085 Microcomputer ..	24.80
80/7	MCS 48 Users Manual 8048 Microcomputer ..	24.80
80/5	Intel Data Katalog 1977/1978 .....	19.80
80/42	6500 Software Manual .....	19.80
80/43	6500 Hardware Manual .....	19.80
80/45	6500 Briefing Notebook .....	9.80
80/46	6500 Datenblattsammlung .....	9.80
80/27	Z 80-Datenbuch .....	49.—
80/28	Z 80-Applikationen .....	14.80
80/29	Z 80-Assemblerhandbuch .....	29.80
80/3	Z 80-Assembler-Manual .....	39.—
80/20	Dr. Dobbs' Journal Vol. 1 .....	59.—
80/48	BASIC Software, Volume VI .....	199.—
80/49	BASIC Software, Volume VII .....	159.—
80/50	BASIC Software Volume I .....	99.—
80/51	BASIC Software Volume II .....	99.—
80/52	BASIC Software Volume III .....	149.—
80/53	BASIC Software Volume IV .....	39.—
80/54	BASIC Software Volume V .....	39.—
80/56	My Computer Likes Me (BASIC-Einf.) .....	9.80
80/57	Computer Games (PCC-Games) in BASIC ....	9.80
1015	Beginners Guide to Microprocessors .....	29.80
1055	BASIC-Cookbook .....	24.80
80/58	Instant BASIC (Einführung BASIC) .....	29.80
80/59	6502 Software Paket .....	89.—
80/60	6800 Software Paket .....	89.—
80/61	8080/8085/8048 Software Paket .....	89.—
80/65	8080/8085 Assembly Lang. Manual .....	29.80
80/66	MCS 85 Product Description .....	9.80
80/67	MCS 48 Product Description .....	9.80
80/68	MCS 48 Appl. Seminar Notebook .....	19.80
80/69	MCS 85 Appl. Seminar Notebook .....	19.80
80/70	Peripheral Design Handbook .....	24.80
80/71	First Book of Kim .....	19.80
80/72	GAME Playing with BASIC .....	24.80
80/85	8048 Assembly Lang. Manual .....	29.80
80/76	6800 Text Editing Syst., Listing m. Cassette	129.—
80/77	6800 Text Processing Syst., List. m. Cassette	160.—
80/81	6800 Micro BASIC PLUS List. m. Cassette ..	89.—
80/79	6800 Mnemonic Assembler, List. m. Cass. ....	129.—
80/80	6800 Disassembler, Listing m. Cassette .....	49.—
80/82	6800 Relocator, Listing mit Cassette .....	49.—
80/83	8080 Text Editing System, Listing .....	99.—
80/84	8080 Text Processing System, Listing .....	129.—
80/90	8080 Assembler Listing+Manual .....	129.—
80/92	Elcomp Microcomputer Magazin, 1 Jahresabbo.	59.—
80/93	Zurückliegende Hefte, 5 Stück nur .....	19.80
574	Beginner's Guide to Computer Progr. ....	39.—
752	Computer Programming Handbook .....	45.—
774	Digital/Logic Electronics Handbook .....	35.—
796	MOSFET Circuits Guide .....	24.80
828	Switching Regulators .....	35.—
836	Opto Electronics Guide-Book .....	24.80
874	Master Handbook of Digital Logic .....	45.—
952	Microproc. Progr. for Hobbyist .....	39.—
1000	57 Practical Programs + Games in BASIC...	35.—
8001	8080 Relocator Listing .....	39.—
8000	9.5 K Basic für 6800 .....	149.—



**TRS-80 Spezial-  
fachzeitschrift**  
Ein Magazin nur für  
den TRS-80 Besitzer.  
Erscheint 6 x pro  
Jahr. Viele wertvolle  
Software, Pro-  
grammiertricks,  
Businesssoftware,  
Texteditor, Mailing  
List u. v. a.  
Best.-Nr. 207 a,  
5 Hefte zum Kennen-  
lernen **DM 49.—**  
Jahres-  
abonnement,  
Best.-Nr. 209  
**DM 99.—**



Seit dem Aufkommen von Mikrocomputern ist es zu einem Alptraum der Funkamateure geworden, sich selbst durch eine automatische Station „überflüssig zu machen“. Das hier vorgestellte KIM-1-Programm erlaubt es, in Verbindung mit einem Funkgerät nebst Modem zumindest in der Betriebsart ASCII-RTTY eine „ganz gemütliche“ Funkverbindung (QSO) ohne menschliches Zutun auszuführen, indem der empfangene Text nach Schlüsselworten geprüft und ein mehr oder weniger passender Text ausgesandt wird. Auch für Nicht-Funkamateure ist das Programm sicher ganz interessant, da es die Möglichkeiten einer formatfreien Texterkennung zeigt.

# Amateurfunk-Automat

Die Betriebsart „Funkfern schreiben“ (RTTY) eignet sich wegen ihrer digitalen Informationsdarstellung besonders gut für die Automatisierung. Das hier beschriebene Programm läßt sich durch Ändern der Ein- und Ausgabe-Routinen auch für Baudot-Funkfern schreiben einsetzen; die im Bild dargestellte Version ist jedoch für ASCII ausgelegt.

Der „ernsthafte“ Funkamateur möge entschuldigen, daß das Programm sicher einen der übelsten Späße darstellt, den man sich derzeit im Rahmen der Mikrocomputer-Verwendung vorstellen kann, und möge es als Entschuldigung werten, daß die Programmentwicklung ebenso wie einige Test-Verbindungen mit dem „OM KIM“ bereits viel Freude bereitet haben. Die Tabelle 1 zeigt einen typischen Verlauf einer solchen Funkverbindung, und es wird sofort klar, daß sie nicht in einer trockenen, formalistischen Computersprache abgewickelt wird, sondern in elementarster Umgangssprache. Trotzdem hofft der Verfasser, daß er ab und zu auch noch mit „normalen Funkamateuren“ in Verbindung treten kann. (Über die postalische Zulässigkeit dieses „Amateurfunk-Automaten“ möge sich Gedanken machen, wer dafür kompetent ist!)

## Hardware-Voraussetzungen

Selbstverständlich muß der Computer die Möglichkeit haben, vom Video-Terminal auf das FSK-Modem für Empfang umzuschalten. Das geschieht hier mit der I/O-Leitung PB 0. Mit PB 5 kann der Sender der Funkstation eingeschaltet werden. Die notwendigen Interface-Schaltungen können sehr einfach sein, richten sich aber nach den Pegelverhältnissen an Modem und Funkgerät. FSK-Modulator und -Demodulator (z. B. aus dem Sonderheft HOBBY-COMPUTER 1) werden mit dem TTY-Aus- und -Eingang des KIM-1 verbunden. Die Übertragungsgeschwindigkeit entspricht derjenigen, mit der beim Initialisieren des Computers das ange-

schlossene Video-Terminal arbeitete, auf dem man empfangene und gesendete Texte mitlesen kann.

## Software

Das Programm ist recht komplex und wird hier auch nicht bis in alle Einzelheiten erläutert. Trotzdem hier einige Angaben über die wichtigsten Adressen. Einige ASCII-Zeichenfolgen lauten:

```
00D4 DC1YB (wird zu Beginn jeder
    Sendung ausgestrahlt)
00DB PSE K (Code für Empfangs-/Send-
    de-Umschaltung)
00E0 CQ DE (Anruf-Code A)
00E6 DC1YB DE (Anruf-Code B)
```

Tabelle 2 enthält die übrigen wichtigen Adressen im Programm. Außer dem Programm selbst wird selbstverständlich ein Textblock benötigt, der die Schlüsselwörter und dazu passende Texte enthält, die als Antwort ausgestrahlt werden sollen. Dieser Textblock beginnt bei 0200 und kann z. B. mit dem Programm „KIM auf Datensuche“ aus FUNKSCHAU 1978, Heft 24, einge-

geben werden. Er hat folgendes Format:

```
CR WORT * WORT # TEXT CR WORT
# TEXT CR WORT * WORT * WORT
# TEXT 00
```

CR ist jeweils ein Carriage-Return-Zeichen (hex 0D). WORT ist das jeweilige Schlüsselwort und TEXT der auszugebende Antworttext. Einem Antworttext lassen sich auch beliebig viele Schlüsselwörter zuordnen. Hinter dem letzten Antwort-Text muß ein Null-Byte stehen, das das Text-Ende angibt. Ein Beispiel:

```
CR REGEN * WETTER * SONNE #
WAS SCHERT MICH DAS WETTER!
CR
```

Der Antworttext „Was schert mich das Wetter!“ wird ausgegeben, wenn wenigstens eines der Schlüsselwörter Regen, Wetter und Sonne erkannt wird.

## Funktionsweise

Hat man ein paar Schlüsselwörter und Antworttexte ab 0200 geladen, so kann man das Programm bei 0000 starten.

**Tabelle 1: Beispiel für ein „automatisches QSO“**

Der vom Benutzer gesendete Text ist am Anfang mit „—“ gekennzeichnet. Für einige Rufzeichen enthält die Texttabelle des Computers auch die Namen (hier „Karl“). Der Gesamtumfang der Texttabelle betrug hier rund 3 KByte, weshalb sie hier nicht abgedruckt wird.

```
—DC1YB DE DC2MPA SERVUS. ALTES HAUS! PSE K
DC1YB OK, KARL! SERVUS AUCH!
—DC1YB DE DC2MPA WIE IST BEI DIR DAS WETTER? HIER IST ES ZIEMLICH
KALT, PSE K
DC1YB OK, KARL! WAS SCHERT MICH DAS WETTER! MEINE TEMPERATUR IST
GANZ ANGENEHM.
—DC1YB DE DC2MPA WAS HAELEST DU VON QSY AUF DB0WB?
DC1YB OK, KARL! LEIDER KANN ICH NICHT AUF EINEN ANDEREN KANAL GEHEN.
ICH HASSE RELAISFUNKSTELLEN.
—DC1YB DE DC2MPA BITTE WARTE, ICH HOLE MIR EIN BIER. PSE K
DC1YB OK, KARL! ICH HABE GENUG ZEIT. ALKOHOL BEKOMMT MIR NICHT.
—DC1YB DE DC3ME DU BIST WOHL EIN KIM-1? PSE K
DC1YB OK, KNU! JA, DER 6502 IST EIN FEINER PROZESSOR.
—DC1YB DE DC2MPA ICH GEHE ESSEN UND MACHE QRT. PSE K
DC1YB OK, KARL! GUTEN APPETIT. SERVUS AUCH!
```

Sobald die Zeichenfolge DC1YB DE oder CQ DE (jeweils mit einem Leer-  
raum hinter DE) erkannt wird, über-  
nimmt das Programm alle nun empfan-  
genen Zeichen (max. 112) in den Buf-  
ferspeicher ab 0100, bis die Zeichen-  
folge PSE K auftritt. PSE K veranlaßt  
den Computer, die Station Senden zu  
schalten, das Modem auszuschalten  
und – damit der Gegenstation etwas  
Umschaltzeit bleibt – einige Sekunden  
zu warten, bevor das Computer-Rufzei-  
chen (DC1YB) ausgestrahlt wird. Dem  
Rufzeichen folgen alle Antworttexte,  
deren Schlüsselworte erkannt wurden.  
Die Suchzeit kann bis zu einigen Se-  
kunden betragen, da das formatfreie  
Suchen recht zeitintensiv ist; die Me-  
thode hat aber den Vorteil, daß auch  
Schlüsselworte innerhalb eines länge-  
ren Wortes erkannt werden (z. B.  
„SCHIFF“ in „DONAUSCHIFF-  
FAHRT“).

Sind alle möglichen Antworttexte  
gesendet, so wird noch ein CRLF und –  
am Beginn der nächsten Zeile – ein sog.  
Prompting-Zeichen ausgegeben, das  
dem Benutzer anzeigt, daß er jetzt wie-  
der dran ist.

Was die Gestaltung der Texttabelle  
angeht, so sei dem Leser selbst die Ent-  
wicklung seiner Optimal-Version über-  
lassen; denn die Texttabelle bestimmt  
im wesentlichen die Leistungsfähigkeit  
des Programms. Es ist ganz reizvoll,  
herauszufinden, welche Antworttexte  
auf häufig vorkommende Schlüssel-  
worte am besten passen.

Letzten Endes spielt dabei auch das  
Lieblings-Konversationsthema des  
Operators eine Rolle; wenn man sich  
auf ein Gebiet beschränkt, kann man  
den Computer zu manchmal recht über-  
raschenden Antworten bringen.

Herwig Feichtinger

```
0000 20 2F 1E JSR 1E2F
0003 A9 3E LDA =3E
0005 20 A0 1E JSR 1EA0
0008 A9 21 LDA =21
000A 8D 03 17 STA 1703
000D A9 01 LDA =01
000F 8D 02 17 STA 1702
0012 A2 00 LDX =00
0014 20 5A 1E JSR 1E5A
0017 D5 E6 CMP E6,X
0019 D0 0A BNE 0025
001B E8 INX
001C E0 09 CPX =09
001E D0 F4 BNE 0014
0020 F0 0C BEQ 002E
0022 20 5A 1E JSR 1E5A
0025 D5 E0 CMP E0,X
0027 D0 E9 BNE 0012
0029 E8 INX
002A E0 06 CPX =06
002C D0 F4 BNE 0022
002E A2 00 LDX =00
0030 E8 INX
0031 86 F3 STX F3
0033 A6 F5 LDX F5
0035 20 5A 1E JSR 1E5A
0038 D5 DB CMP DB,X
003A F0 02 BEQ 003E
003C A2 FF LDX =FF
003E E8 INX
003F 86 F5 STX F5
0041 E0 05 CPX =05
0043 D0 21 BNE 0066
0045 A9 21 LDA =21
0047 8D 02 17 STA 1702
004A A2 12 LDX =12
004C A9 FF LDA =FF
004E 8D 07 17 STA 1707
0051 2C 07 17 BIT 1707
0054 10 FB BPL 0051
0056 CA DEX
0057 D0 F3 BNE 004C
0059 20 2F 1E JSR 1E2F
005C E8 INX
005D B5 D4 LDA D4,X
005F F0 10 BEQ 0071
0061 20 A0 1E JSR 1EA0
0064 10 F6 BPL 005C
0066 A6 F3 LDX F3
0068 9D 00 01 STA 0100,X
006B E0 F0 CPX =F0
```

```
006D D0 C1 BNE 0030
006F F0 BD BEQ 002E
0071 A2 00 LDX =00
0073 BD 01 01 LDA 0101,X
0076 9D 00 01 STA 0100,X
0079 E8 INX
007A E0 F0 CPX =F0
007C D0 F5 BNE 0073
007E C6 F3 DEC F3
0080 D0 03 BNE 0085
0082 4C 00 00 JMP 0000
0085 A9 F0 LDA =F0
0087 85 FA STA FA
0089 A9 01 LDA =01
008B 85 FB STA FB
008D 20 63 1F JSR 1F63
0090 A0 10 LDY =10
0092 B1 FA LDA (FA),Y
0094 F0 DB BEQ 0071
0096 C9 2A CMP =2A
0098 F0 04 BEQ 009E
009A C9 0D CMP =0D
009C D0 EF BNE 008D
009E A2 FF LDX =FF
00A0 C8 INY
00A1 E8 INX
00A2 B1 FA LDA (FA),Y
00A4 C9 2A CMP =2A
00A6 F0 0B BEQ 00B3
00A8 C9 23 CMP =23
00AA F0 07 BEQ 00B3
00AC DD 00 01 CMP 0100,X
00AF D0 DC BNE 008D
00B1 F0 ED BEQ 00A0
00B3 A0 0F LDY =0F
00B5 20 63 1F JSR 1F63
00B8 B1 FA LDA (FA),Y
00BA C9 23 CMP =23
00BC D0 F7 BNE 00B5
00BE 20 9E 1E JSR 1E9E
00C1 A0 0F LDY =0F
00C3 20 63 1F JSR 1F63
00C6 B1 FA LDA (FA),Y
00C8 F0 02 BEQ 00CC
00CA C9 0D CMP =0D
00CC F0 A3 BEQ 0071
00CE 20 A0 1E JSR 1EA0
00D1 18 CLC
00D2 90 ED BCC 00C1
```

```
00D4 44 43 31 59 42 20 00
00DB 50 53 45 20 4B
00E0 43 51 20 44 45 20
00E6 44 43 31 59 42 20 44 45 20
```

**Tabelle 2:**  
**Wichtige Adressen**

0000	Startadresse des Programms
004B	Verzögerung zwischen Sender-Einschalten und Textaussendung
000E	PB-Daten für Empfang
0046	PB-Daten für Senden
0004	„Prompting“-Zeichen
0100...01EF	Empfangs-Buffer
0200...XXXX	Texttabelle
008A	Page des Texttabellen-Beginns – 1

#### KIM-Monitor-Adressen

1E2F	CRLF ausgeben
1E9E	Space ausgeben (X unverändert)
1E5A	Zeichen vom Terminal holen (X unverändert)
1EA0	Zeichen am Terminal ausgeben (X unverändert)
1F63	Pointer FA, FB um 1 erhöhen (A, X, Y unverändert)
1707	Timer im 6530 (ms)
1700	Port A. Datenregister
1701	Port A. Richtungsregister
1702	Port B. Datenregister
1703	Port B. Richtungsregister

#### Wichtige Programmteile

0000	Initialisierung, Prompt-Zeichen ausgeben
0014	Auf „DC1YB DE“ oder „CQ DE“ warten
002E	Eingangsbuffer-Zeiger auf 0100 setzen
0068	Buffer mit den empfangenen Zeichen füllen
0071	Buffer um 1 Zeichen weiterschieben
0038	Auf „PSE K“ warten
0045	Sender einschalten, einige Sekunden Verzögerung
0085	Schlüsselworte der Texttabelle mit dem Buffer vergleichen
00BE	Antworttext ausgeben

#### Eigenschaften der ASCII-Übertragung

<b>Geschwindigkeit</b>	300 Baud, min. 1 Stopbit
<b>Parity-Bit</b>	konstant Null bzw. Zeichen-Invertierung
<b>Modulationsart</b>	F2; H = 2100...2425 Hz, L = 1175...1300 Hz
<b>Sendeformat</b>	2...5 s H-Ton; X DE Y; Text; PSE K X, Y = Rufzeichen (je max. 7 Zeichen, ohne Leerraum vor und hinter der Ziffer)
<b>Steuerzeichen</b>	Line Feed (0A), Carriage Return (0D), Back Space (08), NUL (00, keine Wirkung)
<b>Zeichensatz</b>	Alle ASCII-Zeichen von hex 20 bis hex 7E; Rufzeichen und Systembefehle nur hex 20...5A

#### Stichworte zum Inhalt

KIM-1, 6502, RTTY, Schlüsselwort-Suche, Antworttexte, QSO

Dieses Programm – es umfaßt nicht einmal 1/4 KByte – kann per Funkfern schreiben eine recht unterhaltsame Konversation betreiben



## Microcomputer als Contest-Helfer

Es gehört geradezu zu den klassischen Einsatzmöglichkeiten von Computern, aus einer langen Liste eine bestimmte Information herauszusuchen. Das hier vorgestellte BASIC-Programm (Bild) ist in der Lage, bis zu 1000 Funkverbindungen zu speichern (laufende Nummer, Rufzeichen, Frequenzband) und festzustellen, ob ein bestimmtes Rufzeichen auf einem bestimmten Band schon gearbeitet wurde. In der aufgelisteten Form ist das Programm für ein TRS-80-Microcomputersystem ausgelegt, das mit BASIC Level II und mindestens 16 K RAM ausgerüstet ist.

Da bei einer Betriebsstörung, insbesondere bei Stromausfall, sämtliche im RAM befindlichen Informationen verlorengehen können, besitzt das Programm eine Möglichkeit zur Datensicherung. Nach jeweils fünf neu gearbeiteten Rufzeichen werden die notwendigen Informationen auf die Magnetbandkassette des Systems abgespeichert. Tritt eine Betriebsstörung ein, so können die bis dahin aufgezeichneten Daten wieder in den Computer geladen werden. Die folgende „Gebrauchsanleitung“ verdeutlicht, wie das Programm arbeitet:

1. Programm laden und laufen lassen.

2. Auf die Frage des Computers „ANFANG (1) ODER RESTART (2) ?“ je nach Lage durch Eingabe der Ziffern 1 oder 2 antworten. Mit RESTART ist die Wiederinbetriebnahme des Programmes nach einer Betriebsstörung gemeint, wobei die bis dahin abgespeicherten Daten von der Kassette geladen werden.

Hat man mit 1 geantwortet, geht es wie folgt weiter:

3. Nach der Frage „KASSETTEN-RECORDER AUFNAHMEBEREIT?“ legt man die Kassette in den Recorder ein, auf die die Daten aufgezeichnet werden sollen. Hat man die Aufnahmebereitschaft hergestellt, so läßt man durch Drücken der „Enter“-Taste das Programm weiterlaufen.

4. Auf die Frage „WELCHES BAND (METER) ?“ gibt man die Wellenlänge des Bandes ein, auf dem zuerst gearbeitet wird.

5. Auf die Frage „CALL ?“ kann man das jeweils in Frage kommende Rufzeichen eingeben. Antwortet der Computer mit „CALL OK !!! QSO (J ODER N) ?“ so gibt man ein J oder N ein, je nachdem ob mit der betreffenden Station eine Funkverbindung zustandegekommen ist oder nicht. Antwortet man mit J, so wird das Call in die Liste der gearbeiteten Stationen aufgenommen. Der Computer geht dann zur nächsten QSO-Nummer über.

Stellt man später fest, daß man sich bei der Eingabe eines Rufzeichens geirrt hatte, so kann man um jeweils eine QSO-Nummer zurückrücken, wenn man auf die Frage „CALL ?“ mit der Eingabe eines „Klammeraffen“ (a) antwortet.

Will man auf ein anderes Band überwechseln, so antwortet man auf die Frage „CALL ?“ durch Eingabe von BAND.

Um am Schluß des Contests sicherzustellen, daß auch die letzten Calls noch auf die Kassette geschrieben werden, antwortet man auf die Frage „CALL ?“ mit der Eingabe von END.

Wird nach einer Betriebsstörung ein Restart durchgeführt, so gibt der Computer die notwendigen Anweisungen. Treten beim Laden der Daten Lesefehler auf, so liegt dies möglicherweise an der nicht richtig eingestellten Lautstärke des Kassettenrecorders.

Wie bereits erwähnt, ist das CONTEST-Programm speziell auf das Level II BASIC von TANDY zugeschnitten.

```

10 REM CONTEST - PROGRAMM FUER TRS-80 LEVEL II
20 REM MINDESTENS 16K RAM ERFORDERLICH
30 REM VON GERD DUDDEK (DF5QS). STAND 14.6.79
40 CLS: CLEAR 8000
50 DIM B%(1000), R$(1000)
60 PRINT "***CONTEST-PROGRAMM***": PRINT: PRINT
70 INPUT "ANFANG (1) ODER RESTART (2)"; X
80 IF X=2 THEN GOTO 630
90 IF X=1 THEN GOTO 110
100 PRINT: GOTO 70
110 PRINT: PRINT
120 INPUT "KASSETTEN-RECORDER AUFNAHMEBEREIT "; A$: PRINT
130 PRINT: INPUT "WELCHES BAND (METER) "; B$: PRINT
140 GOSUB 530
150 N=1: K=0
160 PRINT: PRINT N; ". QSO"
170 INPUT "CALL"; C$
180 IF LEFT$(C$,1)="@" THEN 440
190 IF LEFT$(C$,4)="BAND" THEN 440
200 IF LEFT$(C$,3)="END" THEN 490
210 PRINT "BAND : "; B%; "M"
220 FOR I=1 TO N-1
230 IF R$(I) = C$ THEN 370
240 NEXT
250 PRINT "CALL OK!!! ";
260 INPUT "QSO ( J ODER N)"; A$
270 IF A$="N" THEN A$=" ": GOTO 160
280 IF A$(">")="J" THEN GOTO 260

```



```

290 R$(N)=C$ : B%(N)=B% : A$= " "
300 N=N+1 : K=K+1
310 IF K<5 THEN 160
320 PRINT:PRINT "*** DATEN-SICHERUNG QSO-NR. ";N-5;"-";N-1
330 PRINT#-1,N-5,R$(N-5),B%(N-5),N-4,R$(N-4),B%(N-4),N-3,
    R$(N-3),B%(N-3),N-2,R$(N-2),B%(N-2),N-1,R$(N-1),B%(N-1)
340 IF X=99 THEN 810
350 K=0: GOTO160
360 STOP
370 IF B%(I)=B% THEN GOTO 590
380 PRINT " ";
390 PRINT R$(I);"    QSO NR.";I;" AUF";B%(I);"M"
400 GOTO240
410 N=N-1
420 IF K>0 THEN K=K-1: GOTO160
430 K=4: GOTO160
440 PRINT
450 INPUT "NEUES BAND (METER) ";B%
460 GOSUB 530
470 GOTO 160
480 STOP
490 IF K=0 THEN 810
500 N=N+5-K : X=99
510 GOTO 320
520 STOP
530 REM  BAND - KONTROLLE
540 IF B%=2 OR B%=10 OR B%=15 OR B%=20 OR B%=40 OR B%=80
    OR B%=160 THEN RETURN
550 PRINT"DIE FOLGENDEN BANDER SIND MOEGLICH:"
560 PRINT"2M, 10M, 15M, 20M, 40M, 80M UND 160M"
570 PRINT:INPUT "WELCHES BAND (METER)";B%
580 GOTO540
590 REM  CALL IST UNBRAUCHBAR
600 PRINT"*****
    *****"
610 PRINT "    AUF ";B%;"M SCHON GEARBEITET. SCHADE!"
620 PRINT : GOTO 160
630 REM  RESTART NACH SYSTEMZUSAMMENBRUCH
640 PRINT
650 PRINT"RESTART NACH SYSTEMZUSAMMENBRUCH":PRINT
660 PRINT"DATENKASSETTE ZURUECKSOULEN UND PLAY-TAST DRUECKEN !"
670 PRINT: INPUT "WIEVIELE QSO WAREN GEFAHREN";K
680 K=5*INT(K/5)
690 INPUT#-1,Q(1),C$(1),M(1),Q(2),C$(2),M(2),Q(3),C$(3),M(3),
    Q(4),C$(4),M(4), Q(5),C$(5),M(5)
700 FOR I=1 TO 5
710 R$(Q(I))=C$(I)
720 B%(Q(I))=M(I)
730 PRINTQ(I);".QSO MIT ";R$(Q(I));" AUF";B%(Q(I));"M"
740 NEXT I
750 IF Q(5)<K THEN GOTO 690
760 PRINT"LADEN BEENDET !"
770 PRINT:PRINT"SCHALTEN SIE AUF AUFNAHME UND DRUECKEN SIE
    'ENTER'
780 INPUT A$
790 B%=M(5):N=Q(5)+1:K=0
800 GOTO 160
810 END

```

Programmversion für einen TRS-80. Das Programm läßt sich ohne große Schwierigkeiten auch für andere BASIC-Computer anwenden und kann bis zu 1000 Rufzeichen speichern

Will man das Programm für ein anderes System abwandeln, so dürfte das Hauptproblem die Datensicherung sein. Verzichtet man auf die Datensicherung, so kann man im Programm die Zeilen 70...120, 310...330 und 630...800 fortlassen.

Eine erste Bwährungsprobe hat das Programm anlässlich der Teilnahme der Clubstation DL0TB am Contest „Europa-Fieldday“ bestanden, der Anfang Juni 1979 durchgeführt wurde. Dort lief ein TRS-80 im 24-Stunden-Dauerbetrieb, gespeist von einem Diesel-Aggregat. Dabei wurden über 700 Rufzeichen im Computer gespeichert.

#### Stichworte zum Inhalt

Amateurfunk, Contest, TRS-80, BASIC Level II.

## Mikrocomputer im Amateurfunk

Viele Mikrocomputer-Anwender wissen nicht, was sie nach einiger Zeit eingehenden Lernens von Mikroprozessor-Befehlssatz und Programmierkniffen mit ihrer „Intelligenzplatine“ anfangen sollen. Die Funkamateure gehören eindeutig nicht zu dieser bemitleidenswerten Sorte von Menschen (das beweisen auch die in diesem Heft veröffentlichten Amateurfunk-Programme für Mikrocomputer).

Ganz grob lassen sich Anwendungen von Microcomputern im Amateurfunk in zwei Gruppen einteilen: Echtzeit-Steuerungen sowie Datenverarbeitung im Sinne von Rufzeichen-Karteien, Logbuchführung usw. Welcher von beiden Gruppen man mehr Bedeutung beimißt, ist individuell unterschiedlich, bestimmt aber letztlich die Auswahl des am besten geeigneten Mikrocomputer-Systems.

Echtzeit-Anwendungen bedingen eine hohe Verarbeitungsgeschwindigkeit, die sich praktisch nur in der Maschinensprache des Prozessors erreichen läßt – Basic ist zu langsam, da der dazu notwendige Interpreter zu viel Zeit beansprucht. Will man also per Mikrocomputer mit möglichst geringem Hardware-Aufwand funkfern-schreiben oder Morsezeichen decodieren, so sind Computer wie der AIM-65 oder auch der KIM-1 sicher besser geeignet als Basic-Computer, da sie sich komfortabler in Maschinensprache programmieren lassen. Bei solchen Aufgaben wie Logbuchführung, Textverarbeitung, Adressenverwaltung usw. sind dagegen Basic-Computer mit einem Bildschirm-Interface zu bevorzugen.



# Entfernungsberechnung mit QTH-Kennern

Insbesondere bei Funkverbindungen im VHF-, UHF- und SHF-Bereich sind genauer Standort und Entfernung der beteiligten Stationen von besonderem Interesse. Um den Standort ihrer Station in prägnanter Form übermitteln zu können, verwenden die Funkamateure einen besonderen Code, die sogenannten QTH-Kenner. Hierbei handelt es sich gewissermaßen um eine Verschlüsselung der geographischen Koordinaten Länge und Breite. Beispielsweise hat ein Teil der Stadt Münster (Westf.) den QTH-Kenner DL09h, was den Koordinaten 7,63 Grad östlicher Länge und 51,98 Grad nördlicher Breite entspricht.

Sind die QTH-Kenner der Standorte zweier Funkstationen bekannt, so kann man ihre Entfernung berechnen. Dazu muß man zunächst die QTH-Kenner in die Koordinaten zurückverwandeln und kann dann anhand einer Standardformel die Entfernung bestimmen. Da dies allerdings etwas umständlich ist, liegt es nahe, hier einen Heim-Computer einzusetzen.

Im Bild ist ein BASIC-Programm aufgelistet, das diese Rechnungen ausführt. Ein Unterprogramm (ab Zeile 270) bewerkstelligt zunächst die Ermittlung der zu dem jeweiligen QTH-Kenner gehörigen Koordinaten.

Die verwendete Formel trägt der Tatsache Rechnung, daß Computer trigonometrische Funktionen nur mit einer Genauigkeit von etwa 7 Stellen berechnen, es sei denn, man verwendet spezielle Unterprogramme. Die Näherungsformel gilt um so genauer, je geringer die Entfernung ist. Selbst bei Entfernungen von 1000 km ist der Fehler aber praktisch noch kleiner als die Ungenauigkeit, die sich aus der Verwendung der QTH-Kenner zur Standortbestimmung ergibt.

Das Programm ist für das Level-II-BASIC des TRS-80 geschrieben, dürfte aber ohne große Änderungen auch auf anderen Systemen laufen. Voraussetzung ist allerdings das Vorhandensein von gleichwertigen Befehlen zur String-Manipulation. Gerd Duddek

```

10 REM QTHDIS -ENTFERNUNGSBERECHNUNG ANHAND VON QTH-KENNERN
20 REM TRS-80 BASIC LEVEL II - PROGRAMM STAND 18.6.79
30 REM VON GERD DUDDEK (DF5QS)
40 W=.174533 : REM UMRECHNUNGSFAKTOR GRAD/BOGENMASS
50 CLS
60 PRINT"          Q T H D I S"
70 PRINT
80 PRINT"ENTFERNUNGSBERECHNUNG ANHAND VON QTH-KENNERN"
90 PRINT:PRINT
100 INPUT "EIGENER STANDORT (QTH-KENNER)";Q$
110 IF LEN(Q$)<>5 THEN PRINT:GOTO100
120 GOSUB280
130 X1=LG : Y1=BR
150 PRINT
160 C1=COS(W*Y1)
170 PRINT
180 INPUT"STANDORT DER GEGENSTATION (QTH-KENNER)";Q$
190 IF LEN(Q$)<>5 THEN 170
200 GOSUB 280
220 X2=LG : Y2=BR
240 E=111.2*SQR((Y1-Y2)^2+(X1-X2)^2*C1*COS(W*Y2))
250 PRINT"ENTFERNUNG :";INT(E+.5);"KM"
260 GOTO170
270 REM BERECHNUNG VON LAENGE UND BREITE AUS DEM QTH-KENNER
280 L$=MID$(Q$,1,1)
290 LA=ASC(L$)-65
300 M1$=MID$(Q$,3,2)
310 M1$=STR$(VAL(M1$)+99)
320 M1$=MID$(M1$,3,2)
330 LB=VAL(MID$(M1$,2,1))
340 LB= 12*LB
350 RESTORE
360 FOR I=1 TO ASC(MID$(Q$,5,1))-64
370 READ LC,BC
380 NEXT I
390 L1 = 2*LA
400 IF LA<19 THEN 420
410 L1=2*(LA-26)
420 L2=LB+LC+2
430 LG=L1+L2/60 : REM LAENGE IN GRAD
440 REM BREITE
450 BA=ASC(MID$(Q$,2,1))-64
460 BB=VAL(MID$(M1$,1,1))
470 BB=BB*7.5
480 B1=40+BA
490 IF BA<20 THEN 510
500 B1=40+BA-26
510 B2=BB+BC+1.25
520 BR=B1-B2/60 : REM BREITE IN GRAD
522 PRINT"LAENGE:";.01*INT(100*LG+.5);"GRAD , BREITE :";
524 PRINT.01*INT(100*BR+.5);"GRAD"
530 RETURN
540 DATA 4,0,8,0,8,2.5,8,5,4,5,0,5,0,2.5,0,0,4,2.5,4,2.5

```

QTH-Kenner sind eine Art Geheimcode für geographische Länge und Breite. Dieses TRS-80-Programm errechnet die Entfernung zwischen zwei solchen QTH-Kennern





## Software auf Cassette für den PET TRS-80

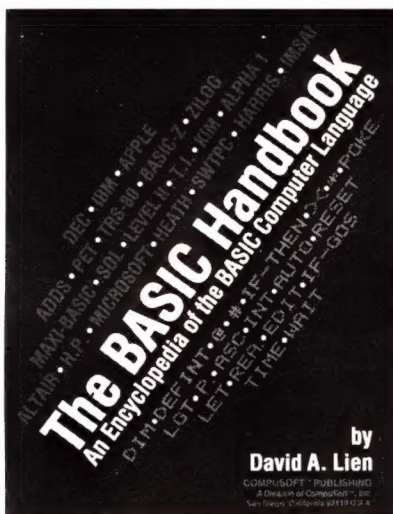
Best.-Nr.	Titel	Preis DM
P 1	U-Bootjagd für PET	49.—
P 2	Einarmiger Bandit (Spielautomat) für PET	29.—
P 3	Black Jack (17 und 4) für PET	49.—
P 4	JOYSTICK mit Programm für PET	149.—
P 5	Biorhythmus für PET	49.—
P 6	Krieg der Sterne (Star Wars) für PET	29.—
P 7	Krieg der Sterne mit Abschießen für PET	29.—
P 8	Las Vegas für PET	29.—
P 9	Haushalts-Utilityprogramme I für PET	29.—
P 10	Haushalts-Utilityprogramme II für PET	29.—
P 11	P 12 Finanzprogramm I+II (Haush.Fin.) f. PET	138.—
P 13	Finanzprogramm III, Immo.Hausk. f. PET	69.—
P 14	Partyprogramm (Party Time) für PET	49.—
P 15	Lernprogramme (Educator I) für PET	29.—
P 15/1	Lernprogramme (Educator II) für PET	39.—
P 16	Musikprogramme für PET	49.—
P 17	Hex-Monitor für den PET	19.80
P 18	Super-Monitor für den PET	19.80
P 19	Musikplattine mit Software f. PET Bausatz	178.—
P 20	Schachprogramm für PET	79.—
P 21	Linear Joystick mit Software Bausatz	199.—
P 22	Programmiertricks für PET	39.—
P 23	BASIC-Kurs für PET	99.—
P 24	Graphik und Bewegung für PET	29.—
P 25	Assembler für PET (2-Pass.)	96.—
P 26	Wortverarbeitungsprogramm (Text) für PET	96.—
P 27	REVERSI für PET	29.80
P 28	Funktionsgenerator mit PET	19.80
P 28/1	Platinenbausatz dazu	79.—
P 29	GAMEPAC I	29.80
P 30	GAMEPAC II	29.80
P 31	Programmierexperimente für PET	39.80
P 32	Stimulating Simulations für PET	49.—
P 33	Astrology Programm für PET	39.80
P 34	Bridge-Programm für PET	49.—
P 35	Computermusik mit dem PET	39.80
P 36	MORSE-TRAINER	29.80
P 37	General Ledger-Hustler 1 (PET 8K)	69.—
P 38	Checking Account (8K PET)	79.—
P 39	Trust Account für Rechtsanwälte (Miete+Hausverwaltung)	69.—
P 40	Legal Diary	69.—
P 41	RENT Accounts	69.—
P 42	Dual Joystick, Fairchild	279.—
P 43	Dual Joysticks, ATARI	279.—
P 44	Music-Box	199.—
P 45	VOICE SET	
	(Spracheingabe und Ausgabe von Tönen)	249.—
P 46	Druckerinterface für RS 232	669.—
	(P 26 Software dazu, Textverarbeitung DM 96.—)	
P 47	Computerspiele	19.80
P 48	Casino und Spielesimulationen	19.80
P 49	Programmieren in Maschinsprache	39.80
P 50	Ein-/Ausgabe Programmierung mit PET	49.80
P 51	Advanced BASIC	69.—
P 52	PILOT Programmierspr. für Anfänger	49.—
P 53	Diät- und Gesundheitsprogramme	19.80
P 54	220 V/50 Hz Schaltinterface (Baus.) m. Softw.	169.—
P 55	Externe Experimentierplatine f. PET m. Softw.	199.—
P 56	Analog Digital / Digital Analog Wandler Platine mit Software	499.—
P 57	Audio-Cassette mit Musik vom PET	9.80
P 58	Cassette mit Computermusik (Audio) Stereo	19.80
P 59	Schalplatte (Stereo) mit Computermusik	19.80
860	57 Programme (auf 3 Cassetten)	
	aus Buch Nr. 31 (8K)	87.—
P 61	JANA-Monitor für PET 2001, 8K	
	Ideal mit Trace und Single Step	29.80
P 62	Joystickprogrammierung	29.80
P 72	Seawolf für PET 2001, 8K	
	ein phantastisches Graphik-Spiel	29.80
204	32 BASIC-Programme für PET (Spiele, Mathematik, Education, Applikation) 5 Cassetten	149.—
8161	Cursor	
	Ein Magazin für PET auf Cassette, 2 Mustercassetten (10 und 11/79)	49.—
<b>Software für KIM-1 auf Cassette</b>		
8110	Musik mit dem KIM	29.80
8111	Starter-KIT für KIM	29.80
8112	Spielprogramme für KIM	29.80
<b>Software für TRS-80 auf Cassette</b>		
TR 1	Stimulating Simulations	39.80
TR 2	Graphic für TRS-80	39.80
TR 3	Microchess für TRS-80	79.—
TR 4	TRS-80 Programmbibliothek (100 Programme) Level II	199.—
TR 5	General Ledger-Hustler 1	69.—
TR 6	General Ledger-Hustler C	89.—
TR 7	Checking Account	79.—
TR 8	Rent Accounts	69.—
TR 9	Legal Diary	69.—
TR 10	Trust Accounts	69.—

Best.-Nr.	Titel	Preis DM
TR 29	Snake Eggs with Sound	
	Bewegungsgraphik mit Ton	49.—
TR 29	ANDROID NIM	
	Eines der phantastischsten Computerspiele, Bewegung und Ton	49.—
TR 30	Lifetwo mit Ton	49.—
TR 31	Cubes	49.—
TR 26	Programmierbarer Tongenerator mit viel Software (Bausatz)	499.—
TR 27	Ein-/Ausgabe Interface für TRS-80 mit Software	549.—
203	Programmbibliothek mit 32 TRS-80-Programmen (5 Cassetten)	149.—

Mailing-List für TRS-80 auf Diskette  
Bestell-Nr. TR 28 ..... 99.—  
Text-80 Wortverarbeitungsprogramm auf Diskette  
Bestell-Nr. TR 29 ..... 99.—  
32 BASIC Programme für TRS-80 auf 5 Cassetten, Mathematik, Graphik, Erziehung, Games, Applications.  
Bestell-Nr. 204 ..... 149.—

**Software auf Cassette für Apple II u. ITT-Computer IT 2020**  
A 006 DATA-Management ..... 78.—  
A 017 Inventur Programm ..... 299.—  
A 014 The BASIC Teacher ..... 84.—  
A 011 Invoicing ..... 189.—  
A 002 Private Sekretärin ..... 189.—  
A 015 Billing Management ..... 299.—  
A 016 Retail Management ..... 189.—  
A 010 Asset Record Program ..... 189.—  
A 007 Programmierte Gymnastik ..... 63.—

**The BASIC Handbook**  
Eine Encyclopedia für BASIC. Eine echte Referenz. Über 50 verschiedene BASIC-Versionen werden behandelt. Beispiele umwandlung. Dieses Buch muß jeder haben, der mit BASIC zu tun hat.  
Bestell-Nr. 219 ..... 49.—



**In letzter Minute eingetroffen:**  
How to Build your own working Robot PET  
238 Seiten, 86 Bilder  
Bestell-Nr. 1141 ..... DM 29.80  
Dr. Dobbs, Vol. IV, Hefte von 1979, 1—5  
Bestell-Nr. 214 ..... DM 49.—  
Dr. Dobbs Jahresabbo, 10 Hefte frei Haus  
Bestell-Nr. 215 ..... DM 129.—  
**Recreational Computing**  
Die erste Personal Computer Fachzeitschrift, 2 Probehefte  
Bestell-Nr. 2162 ..... DM 19.80  
**Creative Computing**  
16 Hefte aus 1977/1978 und 1979, Sonderposten  
Bestell-Nr. 22416 ..... DM 128.—  
The most popular Subroutines in BASIC  
Bestell-Nr. 1050 ..... DM 29.80  
The A to Z Book of Computer Games  
Bestell-Nr. 1062 ..... DM 39.—  
The Best of Creative Computing, Vol. I  
Bestell-Nr. 232 ..... DM 39.—  
The Best of Creative Computing, Vol. II  
Bestell-Nr. 233 ..... DM 39.—  
BASIC Computer Games, David Ahl  
Bestell-Nr. 234 ..... DM 35.—  
Programmieren in Maschinsprache 6502. Einführung und kompletter 2-Pass-Assembler für CBM 16 K (32 K) und PET 8 K. Alles in deutscher Sprache, komplett mit Listing. (Editor, Assembler, Binder und Disassembler).  
Bestell-Nr. 118 ..... DM 98.—  
Programmieren in Maschinsprache Z-80 (TRS-80 — MZ 80-Exidy). Eine Einführung mit vielen Tricks u. 30 Seiten Disassembler-listing.  
Bestell-Nr. 119 ..... DM 49.—  
**CURSOR** Neu! Die Zeitschrift auf Cassette für PET 8 K und CBM. Auf jeder Cassette 5—6 phantastische Programme. Alle zurückliegenden Cassetten Nr. 1—9  
Best.-Nr. 81/7 (Viel gute Software f. wenig Geld) DM 223.—

## HOFACKER

Tegernseer Straße 18  
D-8150 Holzkirchen  
Telefon (08024) 73 31

Lieferung durch den  
Fachhandel oder per  
NN- oder Voraus-  
kasse.  
PSchK. München  
1959 94-807  
oder Scheck.  
Empfohlene  
Verkaufs-  
preise.

## HOFACKER-VERLAG

**Beginners' guide to computer programming**, Brice Ward  
Eine ideale Einführung in die Programmierung von Computern (Mikroprozessoren). Bestell-Nr. 574 ..... DM 39.—

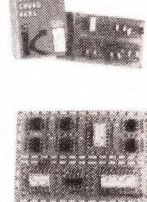


**TTL-Experimentiersatz**. Einführung in die Digitaltechnik, Schaltbeispiele, 5 TTL-IC's, Experimentierplatine, Sockel, Steckmaterial und Anleitungsbuch.  
Bestell-Nr. 105 ..... DM 19.80

**CMOS-Experimentiersatz**, 5 interessante CMOS-Bausteine, Sockel, Experimentierplatine, Anleitungsbuch und Steckmaterial.  
Bestell-Nr. 106 ..... DM 19.80



**Elektronik-Grundlagen-Kurs**, eine leichtverständliche Einführung. Buch mit über 160 Seiten+Platine (Regelschaltung, Sirene, OP-, AMP-Verstärker, Blinkerschaltung).  
Bestell-Nr. 23, 23/1 ..... nur DM 25.70



**Universal Experimentierplatine für IC's** im 40-, 28-, 24-, 16- und 14pol. DIL-Gehäuse, Typ WH 1g. Stabiles Exposé. 210 x 150 mm. Auch für diskrete Bauteile geeignet unter Verwendung von Adaptern. Die Platine eignet sich bestens für alle Versuchsaufbauten in Labor, Schule, Beruf und Hobby. Alle Bauteile und Verbindungen werden gesteckt. Kein Löten mehr! Alle Bauteile bleiben frei von Lötzinn und können immer wieder verwendet werden. Sie sparen Zeit und Geld! Bausatz enthält: Platine verz. und bedruckt, Sockel, Buchsen, Stecker in versch. Farben.  
Bestell-Nr. 41, Typ WH-1g ..... DM 79.—



**Logiktester für TTL, CMOS u. MOS**. Ein praktischer Prüfstift für logische Pegel in Digitalschaltungen. Geringste Stromaufnahme, Anzeige durch zwei LED's. Bausatz komplett mit Gehäuse, Anschlußkabel, Stecker usw. und genauer Bauanleitung. Zusätzlich ein Experimentierheft mit vielen Schaltbeispielen.  
Bausatz, Bestell-Nr. 55 ..... DM 24.80  
Logiktester fertig aufgebaut, Bestell-Nr. 56 ..... DM 29.80

**Adapter zum Aufbau von diskreten Bauteilen**, 5 Stück zusammen im Beutel  
Bestell-Nr. 41/1 ..... DM 14.80



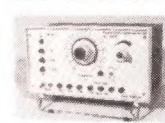
**Kurzlehrgang CMOS-Technik**. CMOS-Einführungskurs. Bestens geeignet für das Selbststudium. Grundlagen, Experimente, viele Schaltbeispiele, Datenangaben. Buch-Nr. 16 mit Experimentier-Platinen-Bausatz WH-1g.  
Bestell-Nr. 16/41 ..... nur DM 98.80

**Kurzlehrgang TTL-Technik**. TTL-Einführungskurs. Der ideale Start in die Digitaltechnik. Genaue Erklärungen, Grundlagen, Einführung und sehr viele Schaltbeispiele.  
Buch-Nr. 21 zusammen mit WH-1g-Experimentierplatine.  
Bestell-Nr. 21/41 ..... DM 98.80  
Beide Kurse zusammen Bestell-Nr. 16/21/41 ..... DM 118.60



**Pultgehäuse**. Formschönes, stabiles Gehäuse für universelle Anwendung, besonders geeignet für Microcomputer (300 x 280 x 130 mm).  
Bestell-Nr. 61 Sonderangebot solange Vorrat ..... DM 19.80

**5 V stabilisiertes Netzteil**. Bausatz mit Trafo, Kabel, Stecker usw. Ausgangsstrom 600 mA.  
Bestell-Nr. 48 ..... nur DM 39.—



**XR 2205 Funktionsgenerator**. Sinus, Dreieck und Rechteckausgang. Bereich 1 Hz bis 100 kHz (0.2 bis 600 kHz. AM- und FM-Modulat. Wobeln, Max. Klirr 1 % (10 Hz bis 10 kHz). Betriebsspannung 12 V, Stromaufnahme 15 mA. XR 2206 Bausatz, Bestell-Nr. 47 ..... DM 69.—

**XR 2206 Bausatz inkl. formschöner Gehäuse**. Frontplatte und allen mechanischen und externen Bauteilen.  
Bestell-Nr. 47/6 ..... DM 199.—  
Gerät fertig montiert, Bestell-Nr. 47/5 ..... DM 249.—





#### A. Osborne Mikrocomputer-Grundwissen

Eine allgemeinverständliche Übermittlung der Kenntnisse als Starthilfe, um dieses faszinierende Gebiet der Technik entdecken und begreifen zu lernen. Keinerlei Vorkenntnisse sind erforderlich. Die Aufmachung ist humorvoll, und nach sechs Lernschritten wird die Materie beherrscht. Durch Testfragen am Ende jedes Kapitels läßt sich der Erfolg kontrollieren und bestätigt sehen.

Preis: DM 36.--\*

#### Einführung in die Mikrocomputer-Technik (2. Auflage)

In diesem Standardwerk führt Adam Osborne den fachlich interessierten Leser weiter in die bahnbrechende Technik der Mikrocomputer-Welt. Der in den USA derzeit wohl am meisten diskutierte Bestseller ist ohne Zweifel die bisher umfassendste, vollständigste und neutralste Darstellung aktuellster Mikrocomputer-Technik. Jetzt in zweiter überarbeiteter Auflage. Hinzugekommen sind Chip-Slice-Produkte und weitere Systembeschreibungen.

Preis: DM 66.--\*

(\*inkl. 6% MwSt., zuzüglich Versandkosten)

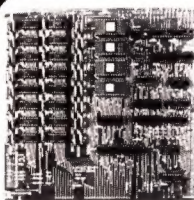
Informieren Sie sich bitte ausführlich, auch über die Osborne-Buchreihe.

**te-wi**

te-wi Verlag GmbH  
Waldfriedhofstr. 30  
8000 München 70

im 2375

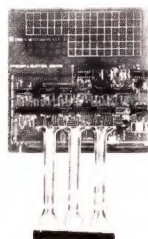
**AKTUELLER  
BUCHTIP!**



#### DYNAMISCHER RAM SPEICHER

8 K/16 K/32 K x 8 (4027 für 8 K - 4116 für 16-32 K)  
4 K EPROM Speichersockel frei (4 x 2708)  
Voll dekodierbar in 4 K Blöcken von H'00 00' - H'F0 00'  
Voll DMA-fähig  
Selbständiger Refresh von Z80-CPU auf NASCOM-1  
Größe: ca. 21 x 21 cm mit 77-poligem Direktsteckverbinder

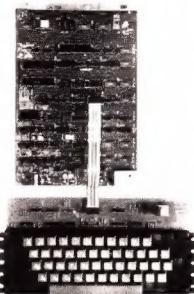
PREIS: (incl. MwSt.) 8 K KIT - DM 441,-  
16 K KIT - DM 734,-  
32 K KIT - DM 1040,-



#### BUFFER BOARD

Volle NASBUS-Pufferung vom NASCOM-1 Board für folgende Erweiterungen:  
Speicherplatinen -  
I/O Platinen -  
8 K ROM Basic Platine -  
Mini-Floppy Controller Platine -  
Graphik-Display Platine usw.  
Alle Erweiterungsplatinen sind ca. 21 x 21 cm und sind mittels NASBUS-Platine miteinander verbunden (bis zu 12 Steckplätze verfügbar).  
Dazu ist ein 19"-Rahmen lieferbar.

Preis: (incl. MwSt.) DM 158.20



#### NASCOM-1 GRUNDSYSTEM (folgendes ist inbegriffen)

Z80 CPU  
2 K RAM  
2 K ROM (1 K NASBUG Monitor, 2 K-Version verfügbar)  
16 x 48 Video-Interface für Standard-T.V. mit UHF-Ausgang  
Tastatur mit 48 Tasten in »QWERTY«-Anordnung (auf 58 Tasten erweiterbar)  
eingebautes Cassetten-Interface  
2 x 8 Bit I/O Ports (PIO) TTL-Kompatibel für Drucker, Relais, Eingänge, Steuerung usw.  
TTY und V24-Schnittstelle, RS232, 20 mA Loop

Preis: (incl. MwSt.) KIT DM 859,-  
SYSTEM DM 1040,-

## NASCOM-1 Computer-System

NASCOM Microcomputers hat sich zur Aufgabe gestellt, ein universelles Micro-Computerkonzept zu verwirklichen, das vom preiswerten Lernsystem bis zum leistungsfähigen Kleincomputer reicht.



#### International NASCOM Microcomputer Club (INMC)

Wir haben diesen Club für NASCOM-Besitzer gegründet, damit ein Austausch von Software unter den Mitgliedern schnell vonstatten gehen kann. Mitglieder bekommen einen Rabatt auf alle weiteren Hardware-Kaufpreise. Nähere Informationen erhalten Sie jetzt von uns auf schriftliche Anfrage.

#### Unser aktueller Hinweis!

8 K Basic auf Kassette, lfb. .... DM 203.40 (inkl. MwSt)  
19 inch Veroframe .... DM 180.80 (inkl. MwSt)  
2K7 Esembler/Editor, lfb. .... DM 180.80 (inkl. MwSt)

#### HERSTELLER

N.A.S. ELEKTRONISCHE HALBLEITER GmbH  
Briener Straße 56, D-8000 München 2  
Tel. 0 89/5 23 31 53/54, Tx. 0 522 061

#### DISTRIBUTOR (Deutschland)

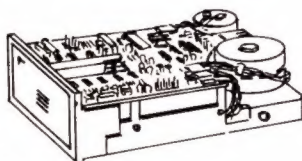
r + r electronic  
Abt. Microcomputer  
Adlerstraße 55, 6900 Heidelberg 1  
Tel. 0 62 21/1 41 29, Telex 0 461 852

# NASCOM-1 Das Z80 Superding

Wir stellen aus zur SYSTEMS, München, Halle 14, OG, Stand 15601B. ELTRO-HOBBY, Stuttgart, Halle 2, Stand 222

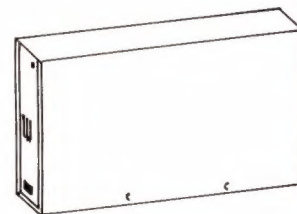
## BASF MINI DISK DRIVE 6106

- Kapazität (uniform.) 125 K bytes
- Transferrate 125 K bits/s
- Shugart kompatibel
- Wir liefern auch die Drives BASF 6101 / 6102 für 8" Disketten, BASF 6104 double headed 8" Disketten und BASF 6108 double headed Mini-Disketten



## Röthahn MINI DISKETTENSTATION

- besteht aus: BASF Mini Disk Drive 6106 und störsicherem Netzteil in stabilem Stahlgehäuse
- mit Controllerkarte lieferbar



Ab Lager lieferbar !

**Röthahn**

Ad. Essich & Co. · Postfach 17 29 · 2900 Oldenburg · Tel. 04.41 / 71 0 71